Universidad
Carlos III de Madrid

Departamento de Ingeniería Informática

TRABAJO FIN DE GRADO

ESTUDIO DE TÉCNICAS DE RESOLUCIÓN PARA EL JUEGO DEL SOKOBAN

Autor: Raquel Herráez Jiménez

Tutor: Carlos Linares López

Leganés, octubre de 2015

Título: Estudio De Técnicas De Resolución Para El Juego Del Sokoban

Autor: Raquel Herráez Jiménez

Director: Carlos Linares López

EL TRIBUNAL

Presidente: Jesús Carretero Pérez

Vocal: Carlos Eduardo Chávez Borges

Secretario: Anabel Fraga Vázquez

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 08 de Octubre de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

"Un programador es la persona considerada experta en ser capaz de sacar, después de innumerables tecleos, una serie infinita de respuestas incomprensibles calculadas con precisión micrométrica a partir de vagas asunciones basadas en discutibles cifras tomadas de documentos inconcluyentes y llevados a cabo con instrumentos de escasa precisión, por personas de fiabilidad dudosa y cuestionable mentalidad con el propósito declarado de molestar y confundir al desesperado e indefenso departamento que tuvo la mala fortuna de pedir la información en primer lugar."

IEEE Grid Newsmagazine

Agradecimientos

Durante estos 4 años de carrera han sido muchas las personas que he conocido y me han aportado pequeños y grandes momentos en esta etapa. Ahora que el grado termina, me encuentro echando la vista atrás y recordando todos esos momentos.

Compañeros de primero con los que el primer día de clase te perdías en el edificio 1 porque el paso a ras de suelo hasta el edificio 7 estaba cerrado por las obras. Compañeros con los que descubrías todo lo relacionado con la Universidad, con los que recorrías las calles de Leganés o con los que te comprabas algo en la cafetería o el hipermercado. Compañeros que ya habían cursado la asignatura y te aconsejaban y ayudaba.

Compañeros nuevos que conocías en segundo porque eran del nuevo turno al que te cambiabas, habían cursado la asignatura el año anterior o porque venían de otra universidad. Compañeros con los que hablabas hasta las cuatro de la mañana porque no lograbais que una parte de la práctica funcionara. Compañeros con los que perdías un poco el contacto porque dejaban la carrera o con los que coincidías menos porque tenían asignaturas pendientes o se quedaron en el otro turno. Compañeros a los que empezabas a conocer más porque te los presentaban otros compañeros y con los que surgía una buena amistad.

Compañeros a los que veías menos porque a partir de tercero tenían otra mención de la carrera pero con los que seguías quedando para comer en la cafetería o ir a tomar pizza. Compañeros de primero a los que no has visto en dos años por no tener asignaturas en común y a los que de repente te encuentras en un aula informática y revivís los viejos tiempos. Compañeros con los muchos días pasas hasta doce horas diarias entre clases y prácticas, de 9 a 21, y a los que ves más que a tu propia familia.

Compañeros que se van durante un año entero de Erasmus a Bélgica o de Movilidad no Europea a Corea del Sur. Compañeros que vienen de Brasil durante un año y con los que enseguida creas una magnífica relación, sorprendiéndote de la cantidad de aficiones compartidas. Compañeros con los que sólo has hablado un par de veces pero que cuando estáis todos reunidos en el aula 4.0.F18 os ayudáis mutuamente y de forma desinteresada para resolver un problema complejo de una práctica.

Todos estos compañeros han sido una parte vital en la carrera. Algunos sólo los conocí por hablar un par de veces y otros se convirtieron en amigos, pero todos son una parte para recordar. Son demasiados nombres para escribirlos, algunos de los cuales no llegué a conocer o se han olvidado con el tiempo, pero hasta el último de ellos se encuentra descrito en los párrafos anteriores. Todos han contribuido a la persona que he sido estos años, y por eso les debo un agradecimiento a estos compañeros.

Pero los compañeros no son las únicas personas a las que les debo tantas cosas. Los profesores que he tenido a los largo de estos años han sido fundamentales en mi formación. Comenzando con Manuel Pereira, el primer profesor que conocí en primero tras encontrar finalmente el camino hasta su clase de Programación en el edificio 7 y terminando con Carlos Linares, el tutor de mi TFG y que fue también mi profesor de la asignatura de Heurística y Optimización. Ahí se encuentran todos los profesores de clase magistral, de clase de problemas, de laboratorios, de prácticas, de tutorías, suplentes…

Todos me han aportado un fragmento del conocimiento y experiencia ganado en estos cuatro años. Algunos solo estuvieron un par de días y otros han llegado a acompañarme durante 3 asignaturas distintas. Ha sido una experiencia muy enriquecedora, pues cada uno tenía sus puntos de vista o sus propias especialidades. Un grupo de profesores de distintas nacionalidades, que aportaban sus experiencias en empresas donde trabajan o que hablaban de aplicaciones de lo estudiado en clase en sus trabajos de investigación. Estos profesores que estaban dispuestos a resolver dudas por correo a casi cualquier hora, a dar tutorías en sus ratos libres o dar clases de repaso para un examen. Porque todos esos esfuerzos han significado mucho para mí.

También me gustaría dar las gracias a mis grupos de amigos, por preocuparse por cómo me iba con mis estudios y por mi estado de ánimo, por obligarme a salir a dar una vuelta para descansar tras semanas de exámenes o prácticas. Sin ellos me habría sido muy difícil desconectar de la rutina y relajarme un poco.

En último lugar, pero que considero el más importante, me gustaría dar las gracias a mi familia, por aguantarme tanto en los buenos como en los malos momentos. Ellos han sabido comprender mi situación de estrés en algunas ocasiones y han soportado mis extraños horarios en los que salía de casa cuando se despertaban mis hermanas y no volvía hasta las 22 de la noche, para ponerme inmediatamente con prácticas de la universidad y apenas poder dedicarlos tiempo. También han tenido que soportar mis rutinas del sueño alteradas y mis cambios de humor y me han apoyado y escuchado cuando más lo necesitaba.

Quisiera dar las gracias a mis hermanos, que en varias ocasiones a lo largo de la carrera me han ayudado probando mis programas y prácticas y de este modo pudiendo encontrar errores en los que yo no había reparado. Ellos son además los mejores amigos que se puede tener y han conseguido hacerme reír cuando más lo he necesitado. No hay mayores amigos que los que te conocen a la perfección y puedes contar con ellos en cualquier momento.

Para finalizar la sección, me gustaría dar un último agradecimiento a mis padres. Ellos me han dado siempre su cariño y apoyo y han puesto sus confianzas en mí, pagándome mis estudios. Sin ellos, nada de esto habría sido posible.

Índice

Índice de Ilustraciones

Índice de Tablas

1 Introducción

El Sokoban es uno de los juegos de tableros más interesantes que existe por el gran abanico de posibilidades que ofrece para los estudios. Es por eso que en este proyecto se tratará de estudiar una pequeña parte de todo lo que abarca este juego.

En este apartado de Introducción se pretende presentar la motivación para la realización de este proyecto, las diferentes etapas que han tenido lugar a lo largo del mismo y un glosario de términos usados a lo largo de este documento. Se terminará detallando la organización empleada para estructurar la presente memoria.

1.1 Motivación

A lo largo de las últimas décadas, la Inteligencia Artificial ha ido ganando una gran cantidad de usos para resolver diversos problemas, tanto en campos de investigación como en la vida cotidiana. Podemos encontrar videojuegos que utilizan Inteligencia Artificial para definir el comportamiento de personajes o realizar el cálculo de óptimo de rutas de movimiento. Encontramos la Inteligencia Artificial también en las predicciones de búsquedas o sugerencias de publicidad al navegar por internet, así como en el análisis de datos. Y en la sociedad actual no nos parece extraño encontrar máquinas que contesten cuando realizamos una llamada telefónica a un servicio técnico.

La Inteligencia Artificial o IA, es una rama de la Informática que suele definirse como la ciencia encargada de dotar de inteligencia a las máquinas o como el estudio y diseño de sistemas inteligentes. Esta disciplina se ha convertido en un elemento muy importante en la actualidad, sobre todo en los campos relacionados con la Industria, la Tecnología y los Negocios. Y es que la IA permite resolver algunos de los problemas más difíciles que existen en los campos de la Informática.

Uno de estos difíciles problemas es el del juego del Sokoban. Este juego se ha demostrado que pertenece a la clase de complejidad PSPACE-Complete [1], que se cree que es una clase de complejidad aun mayor que la del grupo de problemas NP-Completo, a la que pertenecen algunos problemas sobre otros puzles de tablero como es el caso de la consistencia del Buscaminas [2] o el de la solución con menor número de movimientos en el 15-puzle [3].

El Sokoban es un juego por lo general de un único agente o jugador en el que un personaje puede empujar una serie de bloques por el tablero de juego, pudiendo empujar únicamente una caja cada vez. Existen unas casillas especiales, llamadas metas, donde deben colocarse los bloques para completar el juego.

Con esta investigación y desarrollo se pretende implementar distintos algoritmos de búsqueda que sean capaces de encontrar soluciones para este popular juego y analizar cada una de las técnicas empleadas para realizar una comparativa entre ellas.

Además, se pretende realizar también una interfaz gráfica para que los usuarios puedan también jugar a los distintos niveles y visualizar las soluciones halladas por los algoritmos de búsqueda.

Este juego supone además una motivación a nivel personal, ya que el Sokoban es un juego que conocí durante mi primer año de carrera durante la asignatura de programación, y me sirvió para aprender los contenidos básicos de los lenguajes de programación en general y del lenguaje Java en particular.

En aquella práctica final de la asignatura teníamos que implementar el juego en la primera fase como una impresión del tablero por pantalla con algunas funciones y estructuras básicas y en la segunda fase con métodos y clases más avanzados que podían hacer uso de una Interfaz gráfica suministrada por el profesor.

En aquel entonces el juego me encantó en todos los sentidos y sentí algo de pena al no poder aprender en esa ocasión como realizar la parte gráfica con un lenguaje de programación.

Así que este Trabajo Final de Grado ha supuesto para mí la ocasión perfecta para volver al juego que supuso mi iniciación en la carrera, la oportunidad de incluir la parte gráfica que en aquel entonces me parecía algo muy lejano de realizar, y al mismo tiempo una oportunidad para aprender un nuevo lenguaje de programación. Si en aquel entonces mis conocimientos de Java se debían sólo a pequeñas entregas de la asignatura, en esta ocasión el proyecto me supondrá aprender desde cero el lenguaje Python.

1.2 Etapas del Proyecto

Antes de empezar a desarrollar la interfaz gráfica y los algoritmos de resolución mencionados en el apartado anterior, es necesario desarrollar el juego de forma básica, es decir, definir los distintos movimientos posibles y reglas que lo componen y asegurarse de que funciona todo correctamente. Después ya se puede proceder a crear una interfaz más vistosa y a implementar los distintos algoritmos de resolución.

Las etapas del proyecto han sido las siguientes:

- **Etapa 1**: Definición de Objetivos. Se fijaron los objetivos que querían alcanzarse con la realización del proyecto.
- **Etapa 2**: Investigación sobre Python 2 [4], aprender su funcionamiento básico antes de proceder a trabajar con el lenguaje.
- **Etapa 3**: Definición básica del juego del Sokoban. Se definieron en Python las reglas que permiten jugar a un tablero de Sokoban sencillo escrito en un fichero de texto. El tablero representa cada una de las casillas con un carácter ASCII y lo muestra por línea de comandos, solicitando una letra (W, D, X, A) que representa el movimiento realizado. El programa comprueba si el movimiento puede realizarse, en cuyo caso modifica el tablero siguiendo las reglas y lo muestra por pantalla, o indica que el movimiento no se puede realizar (bien porque la tecla no es correcta o el movimiento no está permitido).
- **Etapa 4**: Búsqueda de los niveles a emplear para el estudio. Se procedió a buscar el conjunto de tableros de Sokoban sobre los que se tratará de encontrar soluciones. Se optó por emplear uno de los conjuntos empleados por Andreas Jughanns en su Tesis Doctoral [5], la colección Kids Problems. Este y otros conjuntos suelen encontrarse con un formato concreto de caracteres ASCII para cada casilla, por lo que se adaptaron las reglas para emplear la notación más extendida. El fichero original, que contenía los 61 tableros en un mismo archivo y con comentarios no necesarios para esta tarea, se limpió y se dividió con un pequeño programa Python en 61 ficheros independientes que serán los que se leerán para trabajar sobre cada uno de los niveles.
- **Etapa 5**: Al realizarse la búsqueda de niveles en la etapa anterior y descubrirse las notaciones empleadas, se decidió seguir profundizando en conocer los nombres de las notaciones y conocer más aspectos sobre el dominio del juego del Sokoban, las noticiones utilizadas, estudios existentes y se desarrolló el Estado del Arte del Documento. También se realizó el apartado de *Presupuesto Inicial*.

- **Etapa 6**: Juego en modo gráfico. Tras un estudio sobre programación en Pygame [6] y algo de práctica, se procedió a implementar la primera representación gráfica del juego, definiendo los bucles de juego, los eventos de lectura de teclas y las distintas funcionalidades que emplean las reglas desarrolladas anteriormente.
- **Etapa 7**: Diseño del solucionador. En esta etapa se implementaron los distintos algoritmos de búsqueda que se emplearán para encontrar soluciones, y se fueron aplicando diversas mejoras sobre los mismos para reducir al mínimo los tiempos para encontrar soluciones y la cantidad de nodos generados y expandidos.
- **Etapa 8**: Menú completo. Como parte final de la implementación, se generó un menú completo para la Interfaz Gráfica que permite jugar con los tableros o mostrar las soluciones encontradas para los 61 niveles, escogiendo el algoritmo que se desea mostrar.
- **Etapa 9**: Redacción del resto de apartados de la memoria.

1.3 Glosario de Términos

- **Deathlock:** situación que se produce en una partida de Sokoban cuando es imposible completar el juego realizando ninguna de las acciones permitidas por las reglas.
- **IA:** son las siglas de Inteligencia Artificial. Haces referencia a la rama de la Informática que estudia la creación de sistemas que resuelven problemas de forma autónoma.
- **Kids Problems**: nombre común del conjunto de tableros de Sokoban "Dimitri and Yorick", creado por Jacques Duthen. Consta de un total de 61 tableros de Sokoban y fue realizado pensando en niños de entre 4 y 8 años.
- **Notación XSB**: formato de representación de los tableros de Sokoban mediante una combinación concreta de caracteres ASCII.
- **Notación LURD**: es el formato más extendido para la representación de movimientos en soluciones de Sokoban.
- **NumPy**: librería de Python que permite trabajar con datos, números y arrays.
- **Pygame**: librería para Python que ayuda en el diseño de gráficos para juegos e interfaces gráficas en este lenguaje de programación.
- **Python**: lenguaje de programación interpretado que tiene una filosofía en la que se desea que el código sea legible y donde las indentaciones tienen mucha más importancia que en otros lenguajes de programación.

1.4 Organización del Documento

La estructura del documento será la siguiente:

1. **Introducción:** En este apartado, en el que nos encontramos actualmente, se da una primera visión del documento, hablando de la motivación tras la realización del trabajo. También se definen las distintas etapas del proyecto, los términos más importantes y la organización que se empleará en el presente documento.
2. **Estado del arte:** en la segunda sección de la memoria se introducirá la situación relacionada con el entorno del problema que se plantea, hablando del juego así como de distintas técnicas de resolución empleadas en la Inteligencia Artificial.
3. **Objetivos:** este capítulo expondrá, como su nombre indica, los objetivos que se espera alcanzar mediante la realización del proyecto.
4. **Desarrollo:** este apartado, que supone el núcleo del trabajo, tratará todos los pasos realizados en este proyecto, hablando de distintas versiones empleadas, modificaciones realizadas y decisiones tomadas. Se hablará de la parte encargada de la resolución de los tableros, así como de la interfaz gráfica del juego.
5. **Pruebas y Resultados:** en el capítulo de pruebas se comprobarán los distintos algoritmos empleados y se analizarán los resultados obtenidos a partir de los mismos.
6. **Presupuesto:** está sección recogerá los distintos costes que se derivan de la realización del presente proyecto.
7. **Conclusiones:** se trata de un apartado en el que se recogerán los datos extraídos de la totalidad del trabajo y se comprobará si se han alcanzado los objetivos previstos.
8. **Líneas Futuras:** se mencionarán posibles mejoras o modificaciones que podrían realizarse en un futuro a los pasos realizados en este estudio.

Para terminar el documento incluirá unos apartados de Anexos y Bibliografía, que contendrán información adicional sobre los contenidos tratados y mostrarán las fuentes de las que se ha obtenido información, respectivamente.

2 Estado del Arte

Antes de comenzar a desarrollar el proyecto, es necesario comprender las bases del juego del Sokoban, en el que se centra el siguiente trabajo, así como realizar un estudio de la situación en la que se encuentra todo el entorno que está relacionado directa o indirectamente con el tema a tratar. Por tanto, en este capítulo se definirá el juego, se hablará de distintos métodos de búsqueda, y se procederá a explicar otros estudios del mismo ámbito, así como avances logrados o posibles en un futuro.

2.1 Juego del Sokoban

EL objetivo de esta sección será la de tratar en profundidad el tema del Sokoban, desde su historia, propiedades de las casillas, reglas, bloqueos y distintas modalidades de juego. Además, se hablará también de las notaciones más empleadas para representación de niveles y soluciones.

2.1.1 Introducción

El Sokoban es un popular juego de lógica que consiste en un tablero con distintos tipos de casillas. El jugador debe recorrer el tablero moviendo distintas cajas o piedras hasta unas zonas especialmente diseñadas para este fin. El juego termina cuando todas las cajas están colocadas sobre las metas.

2.1.2 Origen

Sokoban, que significa “Guardián del Almacén” en japonés, es un juego creado por Hiroyuki Imabayashi en 1981 y publicado por su compañía, Thinking Rabbit [7] en un pack de 20 niveles lanzado para NEC PC-8801 en el año 1982.

A continuación se muestra la distribución del primer nivel de esta colección original lanzada por Thinking Rabbit:

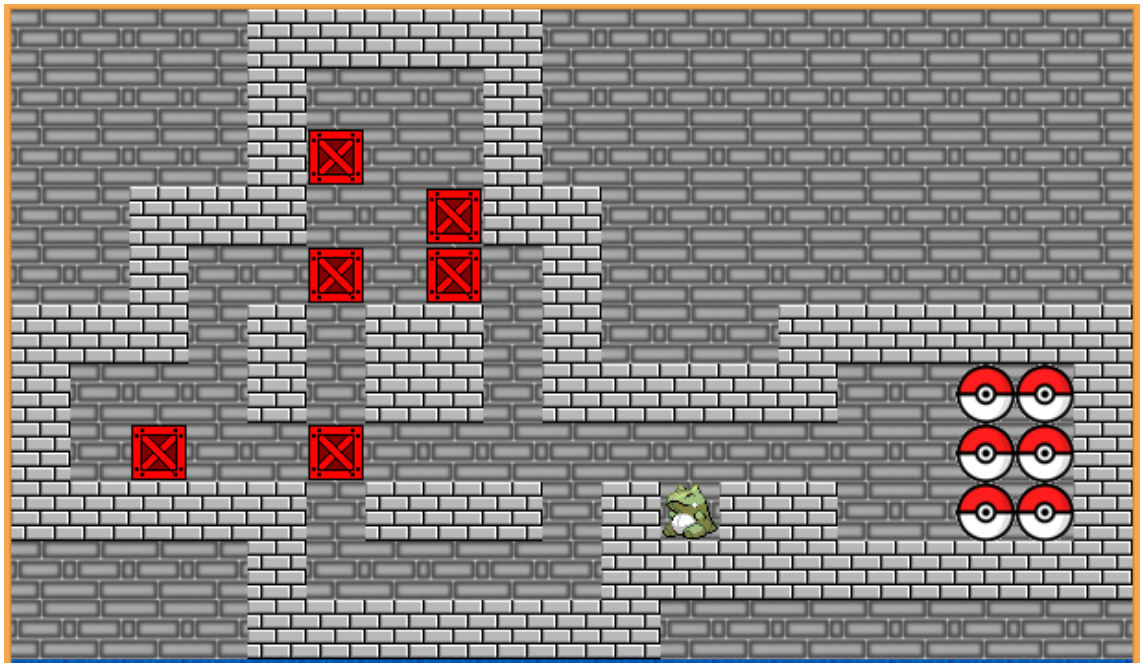

Ilustración 1: Nivel de Sokoban

2.1.3 Tipos de Casillas

Un nivel de Sokoban está compuesto por un conjunto de casillas de distintos tiempos. Cada una de estas casillas tiene unas propiedades especiales:

- **Jugador:** El encargado del almacén. Puede desplazarse por las casillas libres y por las bases, además de empujar cajas.
- **Cajas:** Son objetos distribuidos por el tablero y que deben llevarse hasta las bases. Las cajas o piedras pueden desplazarse por el tablero cuando el encargado las empuja, pero solo pueden estar sobre casillas vacías o metas.
- **Metas:** son las casillas diseñadas para que se coloquen las cajas sobre ellas. Son caminables por el jugador.
- **Muros:** Definen los límites del tablero y no pueden atravesarse por ningún objeto.
- **Casillas Vacías:** casillas por las que el jugador puede desplazarse con normalidad y empujar cajas.

Además, para efectos prácticos y de diseño, cuando un jugador o una caja se encuentran sobre una meta, representan cada uno otro tipo de casilla.

2.1.4 Reglas

En este estudio se tratará únicamente el modo de juego original del Sokoban, ya que existen otras versiones con reglas ligeramente diferentes. Las reglas del juego son bastante sencillas y se reducen a un pequeño conjunto:

- El jugador puede moverse sobre casillas vacías o metas.
- El jugador puede empujar cajas, pero no tirar de ellas.
- El jugador sólo puede moverse en horizontal y vertical, no en diagonal.
- El jugador no puede atravesar cajas ni situarse sobre ellas.
- Las paredes no pueden ser atravesadas ni por el jugador ni por cajas.
- Las cajas pueden encontrarse sobre casillas vacías o bases.
- El jugador no puede empujar más de una caja a la vez.

2.1.5 Deadlocks

Un Deadlock o Bloqueo es una situación del tablero del Sokoban en la que es imposible ganar la partida, ya que las piedras han sido movidas de tal manera que al menos una de ellas no puede ser llevada a una base. Una vez que se llega a una situación de Deadlock, no hay forma de salir de dicha situación mediante los movimientos permitidos en el juego. La única solución para salir del bloqueo es reiniciar la partida, aunque hay versiones del juego que permiten deshacer los movimientos realizados.

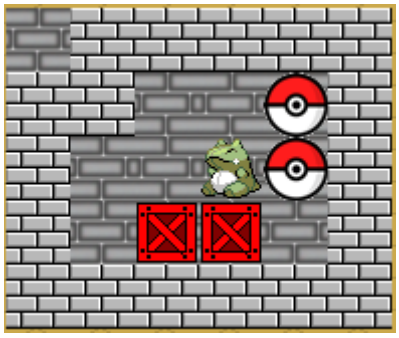

Tabla 1: Ejemplo de Deathlock

2.1.6 Modalidades

Desde el lanzamiento en 1982 de los primeros 20 rompecabezas por Thinking Rabbits, han sido muchas las versiones del juego que han salido al mercado. La propia compañía creadora ha lanzado varias colecciones de problemas a lo largo de los años:

- Sokoban (1982) con 20 puzles
- Sokoban 2 (1984) con 50 niveles
- Sokoban Perfect (1989) con un total de 306 rompecabezas
- Sokoban Revenge (1991) con 306 niveles

Si tenemos en cuenta todos los distribuidores del juego, todas las plataformas para las que ha sido lanzado (PC, Consolas, Móviles…) o todas las colecciones de niveles creadas, las posibilidades son casi ilimitadas. Y es que la creación de niveles de Sokoban es considerado prácticamente un arte. Encontrar representaciones simétricas, bonitas, con ciertas formas, que supongan un reto, que traten de probar cierta estrategia de juego… todo tipo de combinaciones.

Es por esto que con modalidades del juego nos referiremos a variantes del juego original en el que las reglas son algo distintas a las especificadas en el *apartado 2.1.4*. Las principales modificaciones existentes son las siguientes [8]:

- **Modificación de la forma de las casillas**: en lugar de usarse casillas de forma cuadrangular, se emplean otras figuras geométricas, como triángulos (*Trioban*) o hexágonos (*Hexoban*)
- **Modificación del número de jugadores**: en este caso hay más de un encargado del almacén, dándose el conocido como *Multiban*.
- **Modificación de las metas**: empleo de colores (*Block-o-Mania*) o números (*SokoMind Plus*) son algunas de las técnicas más empleadas en estos casos, donde las cajas sólo pueden ir en las metas con el mismo color o número. Existen versiones en las que las cajas deben ser desplazadas a las bases en un orden concreto, como en el caso de *Cyberbox*. También hay modalidades en las que no hay casillas de meta, sino que hay que agrupar cajas del mismo color (*Interlock* y *Sokolor*).
- **Modificación de las acciones del personaje**: en *Pukoban* se permite tirar de las cajas además de empujarlas.
- **Tipos adicionales de casillas**: una modificación que se puede dar en cualquiera de las modalidades de juego y que añade elementos como teletransportes, caminos unidireccionales o agujeros.
- **Modo Inverso**: se parte de la solución y se trata de llegar al estado inicial.

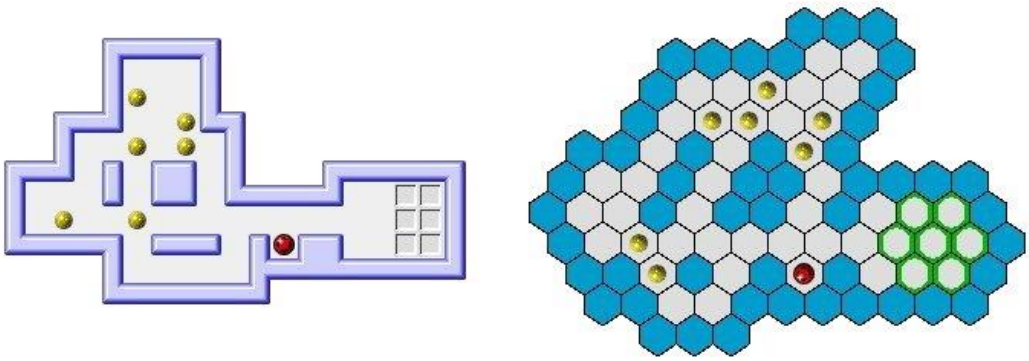

Ilustración 2: Transformación del Nivel 1 clásico a su equivalente en Hexoban

2.1.7 Notación XSB para niveles

La notación XSB es la más extendida para representar los distintos tableros de Sokoban [9]. Esta notación representa cada casilla del juego con un carácter ASCII. La correspondencia entre caracteres y casillas es la que puede observarse a continuación:

Casilla	Carácter
Jugador	@
Jugador sobre meta	+
Caja	$
Caja sobre meta	*
Meta	.
Casilla libre	‘ ‘ (espacio) / - / _
Muro	#

Tabla 2: Notación XSB

El carácter empleado desde los inicios del estándar para las casillas vacías ha sido el espacio (‘ ‘), pero debido a que algunos correos o sitios web eliminan varios espacios consecutivos, es común emplear los guiones (- y _) para representar estas casillas. En archivos de texto el carácter más extendido para representar las celdas de suelo vacío sigue siendo el espacio.

2.1.8 Notación LURD para soluciones

La notación LURD es la más empleada para representar soluciones de tableros de Sokoban. Cada una de las letras indica la dirección en la que se ha movido el jugador, representada por la primera letra del nombre del movimiento en inglés. Además, se distinguen dos tipos de movimientos del jugador en cada dirección: movimiento en el que el jugador se desplaza a una casilla vacía o movimiento en el que el jugador empuja una caja. Los movimientos a una casilla se representan con una letra minúscula, mientras que los movimientos que implican empujar una caja son mostrados con la letra en mayúscula [9].

A continuación se muestra una tabla que recoge todas las representaciones de movimientos:

Movimiento	Empuja Caja	Representación
Izquierda	No	l
	Sí	L
Arriba	No	u
	Sí	U
Derecha	No	r
	Sí	R
Abajo	No	d
	Sí	D

Tabla 3: Notación LURD

2.1.9 Importancia para la Investigación

En 1997 Culberson demostró que el juego del Sokoban es PSPACE-Complete [1]. Esto implica que el juego cumple las características propias de los problemas pertenecientes a esta clase de complejidad computacional. Los problemas PSPACE-Completo son los más difíciles de entre los que se encuentran en la clase PSPACE, ya que encontrar una solución para un problema de la clase PSPACE-Completo supone encontrar una solución para todos los problemas de la clase PSPACE.

La clase PSPACE se define como el conjunto de problemas que son decidibles en espacio polinómico por una Máquina de Turing Determinista [10]. Por su parte, los problemas PSPACE-Completo se definen como aquellos problemas que pertenecen a la clase PSPACE y que cumplen que todos los problemas de la clase PSPACE son transformables a ellos en tiempo polinómico. O lo que es lo mismo, los problemas que pertenecen a PSPACE y que se cumple que existe otro problema PSPACE-Completo reducible al problema en cuestión.

A continuación se incluye una imagen que muestra la relación entre las clases de complejidades más conocidas.

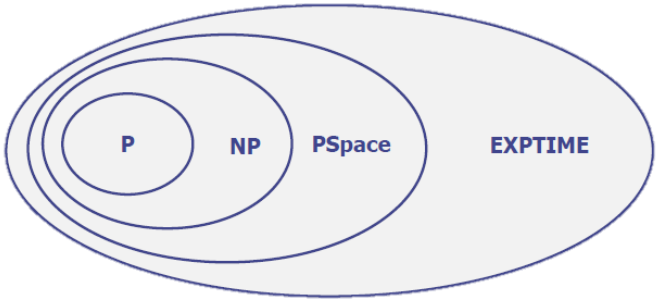

Ilustración 3: Clases de Complejidad

Se conoce que en esta relación de clases se cumple que $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$ y que además se verifica que $P \subsetneq \text{EXPTIME}$, por lo que al menos una de las otras relaciones entre los conjuntos es de inclusión propia, es decir, que al menos uno de los pare P-NP, NP-PSPACE o PSPACE-EXPTIME son distintos.

De este modo, el problema de saber si P=PSPACE, al igual que ocurre con el famoso problema sobre si P=NP, es un problema matemático no resuelto.

Otra relación interesante es la de las clases NP-Completo y PSPACE-Completo, ya que de encontrarse una igualdad entre estas clases, se podrían realizar reducciones entre los problemas de las mismas, algunos de la importancia de SAT (NP-Completo) y Planning (PSPACE-Completo).

Sokoban en concreto pertenece a la clase de complejidad PSPACE-Completo y se sabe que también se encuentra en NP-HARD [11], pero no se sabe cómo se relacionaría esta situación con NP y NP-Completo, ya que la clase de complejidad es la intersección de NP y NP-Hard, pero no está demostrado que todos los problemas de NP-Hard se encuentren en NP-Completo (y por tanto en NP).

A partir de todo esto puedes extraerse conclusiones bastante interesantes sobre el Sokoban. Si Sokoban se pude reducir a algún problema de la clase PSPACE en tiempo polinómico, como Sokoban es un problema PSPACE-Complete, por la definición de este grupo de problemas, se estaría concluyendo que ese problema sería también PSPACE-Complete, por lo que el Sokoban puede emplearse para demostrar si un problema es PSPACE-Complete.

Aún hay más. Como todos los problemas de la clase PSCPACE-Completo tienen la misma complejidad, si se encontrara una solución sencilla para Sokoban que permita dar una respuesta en tiempo polinómico, entonces se habría encontrado esta solución para todos los problemas de su misma clase, pudiéndose concluir que P=PSPACE.

2.2 Algoritmos de Búsqueda

Los algoritmos de búsqueda tienen como finalidad encontrar un elemento que cumple unas características entre todos los estados que componen un espacio de búsqueda. Estas búsquedas pueden contar o no con información del entorno, y además esta información puede ser total o parcial. Según la cantidad de información que se posea del entorno, podemos clasificar los algoritmos de búsqueda en tres categorías:

Cantidad de Información	Tipo de Búsqueda
Ninguna	Búsqueda no Informada
Parcial	Búsqueda Informada
Total	Algoritmo Exacto

Tabla 4: Tipos de Búsquedas

A continuación se explicarán los principales algoritmos de Búsqueda Informada y Búsqueda no Informada, señalando sus principales características e incluyendo el algoritmo en pseudocódigo.

2.2.1 Búsqueda no Informada

La búsqueda no informada es un tipo de búsqueda en la que no se cuenta con ninguna información a priori sobre el espacio de búsqueda. En este caso el espacio de estados debe ser recorrido de forma exhaustiva siguiendo algún patrón de búsqueda.

2.2.1.1 Búsqueda en Amplitud o Breadth First Search (BFS)

La Búsqueda en Amplitud es un tipo de búsqueda no informada en la que los nodos del espacio de estados son expandidos por niveles, de izquierda a derecha. Hasta que todos los nodos de un mismo nivel no han sido expandidos, no se procede a expandir el siguiente nivel [12].

Características:

- **Completitud:** encuentra solución si existe y el factor de ramificación es finito.
- **Optimalidad:** es óptimo si el coste de los operadores es el mismo.
- **Eficiencia:** buena para metas cercanas.
- **Desventajas:** alto consumo de memoria.

```
Amplitud (inicial):
        insertarPrincipio(abierta, inicial)
        éxito = False
        Mientras (abierta!=vacia) y no exito:
                c = extraerPrimero(abierta)
                si meta(c):
                        exito = True
                En caso contrario:
                        generarSucesores(c)
                        para cada sucesor n:
                                insertarFinal(abierta,n)
```

Tabla 5: Pseudocódigo de Amplitud

2.2.1.2 Búsqueda en Profundidad o Depth First Search (DFS)

En la Búsqueda en Profundidad, los nodos se van expandiendo hasta llegar a la profundidad máxima. Si no se encuentra una solución al llegar a esta profundidad, realiza Backtracking, es decir, vuelve al nodo anterior y escoge el siguiente descendiente de ese nodo para continuar la búsqueda [12].

Características:

- **Completitud:** no se asegura que se encuentren soluciones, porque puede caer en ciclos.
- **Optimalidad:** Al no ser completo, tampoco es óptimo.
- **Eficiencia:** bajo consumo de memoria y mejor para metas alejadas.
- **Desventajas:** funciona mal cuando hay ciclos.

```
Profundidad (inicial):
        insertarPrincipio(abierta, inicial)
        exito = False
        Mientras (abierta!=vacia) y no exito:
                c = extraerPrimero(abierta)
                Si meta(c):
                        exito = True
                En caso contrario:
                        generarSucesores(c)
                        para cada sucesor n:
                                insertarPrincipio(abierta, n)
```

Tabla 6: Pseudocódigo de Profundidad

2.2.1.3 Búsqueda en Profundidad Iterativa o Iterative Deepening Depth First Search (IDDFS)

Este tipo de búsqueda utiliza como base la búsqueda en profundidad limitada (Depth Limited Search, DLS), es decir una búsqueda en profundidad que se ejecuta sólo hasta una cierta profundidad. En cada iteración se realiza la búsqueda en profundidad limitada, aumentando en una cantidad la profundidad que se empleará en la siguiente iteración del algoritmo. De este modo, se combinan las ventajas de BFS y DFS [12].

Características:

- **Completitud:** se asegura la completitud.
- **Optimalidad:** es óptimo si el incremento de la profundidad se realiza en intervalos de una unidad.
- **Eficiencia:** tiempo algo mayor al de amplitud.
- **Desventajas:** puede generar muchos estados repetidos.

```
ProfundidadIterativa (inicial):
        profMax = Incremento
        exito = False
        mientras éxito == False:
                exito = ProfundidadLimitada(inicial, profMax)
                profMax += Incremento

ProfundidadLimitada (inicial, profMax):
        insertarPrincipio(abierta, inicial)
        Mientras (abierta!=vacia) y no EXITO:
                c = extraerPrimero(abierta)
                Si meta(c):
                        EXITO = True
                En caso contrario:
                        Si profundidad (c) es menor que profMax:
                                generarSucesores(c)
                                para cada sucesor n:
                                        insertarPrincipio(abierta, n)
```

Tabla 7: Pseudocódigo de Profundidad Iterativa

2.2.2 Búsqueda Informada

En este tipo de búsqueda se cuenta con cierta información del entorno, que se emplea para que los algoritmos de búsqueda visiten primero los nodos más prometedores.

Con esta información que se tiene es posible definir una función heurística que aproxime el coste de llegar desde un estado concreto hasta la meta. Esta heurística suele representarse como h(n), y se calcula para cada nodo del árbol.

Por otra parte se definen además las heurísticas admisibles, que son las que nunca sobreestiman el coste real de encontrar la solución. El empleo de una heurística admisible en algoritmos como A* e IDA* garantizará encontrar soluciones óptimas.

*2.2.2.1 A**

El algoritmo A* es un método de búsqueda informada en el que los nodos sucesores son visitados según el coste de una función calculada para cada estado. Esta función, f(n) se obtiene de la suma de g(n), el coste real hasta ese estado, y h(n), la heurística calculada en ese estado para llegar al nodo meta. De este modo, el algoritmo visitará primero los nodos con menor valor de f(n) por ser los más prometedores [12].

Características:

- **Completitud:** si hay solución la encuentra.
- **Optimalidad:** es óptimo si ramificación finita, todos los costes son mayores que 0 y la heurística empleada es admisible.
- **Eficiencia:** depende de la heurística empleada.
- **Desventajas:** alto consumo de memoria.

```
A* (inicial):
        insertarPrincipio(abierta, inicial)
        cerrada = vacia
        éxito = False
        Mientras (abierta!=vacia) y no exito:
                c = extraerPrimero(abierta)
                si meta(c):
                        exito = True
                En caso contrario:
                        insertarFinal(cerrada, c)
                        generarSucesores(c)
                        para cada sucesor n no predecesor de c:
                                si ya en cerrada o abierta, decidir con cual quedarse
                                insertarFinal(abierta,n)
                                ordenar(abierta, f)
```

*Tabla 8: Pseudocódigo de A**

*2.2.2.2 IDA**

En el algoritmo de IDA*, al igual que en el algoritmo IDDFS, se ejecuta un algoritmo en profundidad limitada con un incremento de la profundidad en cada iteración, pero en este caso en lugar de emplearse la profundidad como parámetro para detener la iteración, se emplea la función heurística. Se fija la condición de parada al valor de la función f(n) del nodo raíz, se expanden todos los nodos que no sobrepasen ese valor, y cuando no se puede continuar, se fija el valor de parada al del f(n) que se ha pasado con menor margen [12].

El valor de f(n) se calcula del mismo modo que en A*, esto es, f(n) = g(n) + h(n)

Características:

- **Completitud:** es completo.
- **Optimalidad:** es admisible y encontrará la solución óptima.
- **Eficiencia:** complejidad de tiempo exponencial pero complejidad espacial lineal, menor que en A*.

```
IDA* (inicial):
        heuMax = h(inicial)
        exito = False
        mientras éxito == False:
                exito = ProfundidadLimitada(inicial, heuMax)
                heuMax = h minimo en pasarse de heuristica

ProfundidadLimitada (inicial, heuMax):
        insertarPrincipio(abierta, inicial)
        Mientras (abierta!=vacia) y no EXITO:
                c = extraerPrimero(abierta)
                Si meta(c):
                        EXITO = True
                En caso contrario:
                        Si profundidad (c) es menor que heuMax:
                                generarSucesores(c)
                                para cada sucesor n:
                                        insertarPrincipio(abierta, n)
```

*Tabla 9: Pseudocódigo de IDA**

3 Objetivos

Entre los objetivos personales que se pretenden alcanzar durante la realización del trabajo, se encuentran el de conocer un lenguaje de programación nuevo y profundizar en él, siendo Python el lenguaje escogido finalmente en este caso. Durante una de las asignaturas del curso se propuso una práctica opcional en Python, un lenguaje que no empleado antes durante el curso, y tras buscar la información básica necesaria para la práctica opcional, decidí aprender más sobre este lenguaje. Por otra parte, durante la carrera se trata poco la visualización de elementos gráficos al programar en algún lenguaje, así que con este proyecto se pretende también adquirir conocimientos para mostrar interfaces y elementos gráficos escritos en el lenguaje Python.

Como objetivos específicos que se pretenden alcanzar en el transcurso del proyecto, se encuentran los siguientes:

- Elaboración de un juego de Sokoban en el lenguaje de programación Python.
- Implementación de distintas técnicas de inteligencia artificial que sean capaces de resolver diferentes tableros del juego.
- Medir la calidad e idoneidad de cada una de las técnicas empleadas, en bases a distintos parámetros medidos, como son:
    - Capacidad o no de solucionar cada uno de los distintos tableros
    - Número de pasos del jugador
    - Movimientos de cajas empleados
    - Tiempo empleado para encontrar la solución
    - Nodos generados/expandidos durante la búsqueda
    - Utilización de espacio en memoria
    - Optimalidad o no de las soluciones
    - Etc.
- Obtención de soluciones que consistan en una serie de pasos para llegar desde el estado inicial del problema a un estado meta.
- Desarrollo de una interfaz gráfica en Python que permita jugar partidas o mostrar las soluciones encontradas.

4 Desarrollo

4.1 Decisiones Tomadas

A la hora de implementar los distintos algoritmos, la primera decisión que había que tomar era el lenguaje de programación a utilizar. Se optó por emplear Python, por ser un lenguaje que no se emplea en ninguna práctica obligatoria de la carrera y del que yo no tenía conocimientos. El lenguaje llamó mi atención cuando en una asignatura del grado se propusieron varias prácticas optativas, una de las cuales utilizaba Python de forma sencilla. Por aquel entonces ya estaba planeando el lenguaje a emplear en el proyecto, teniendo como principales candidatos los lenguajes C++ (del que tengo conocimientos básicos) y Python (del que no conocía nada). Aproveché la ocasión para conocer el lenguaje y me gustó bastante, así que me decidí por él.

Dentro de Python había otra decisión que tomar, y era la de emplear Python 2 o Python 3. Tras investigar un poco se optó por Python 2, ya que es la versión más documentada y extendida, aunque Python 3 cada vez gana más fuerza y está también bastante extendido.

Por otra parte, se decidió emplear la librería Numpy para facilitar la lectura de datos de fichero con un determinado formato y para trabajar fácilmente con ellos en una matriz bidimensional que represente el tablero de juego.

La siguiente decisión era elegir el método para desarrollar la interfaz gráfica del juego, optándose por emplear Pygame. Este conjunto de módulos para Python está basado en la librería SDL y permite crear juegos y programas multimedia [6].

A la hora de ejecutar los distintos algoritmos, como puede que algunos no encuentren soluciones o tarden un tiempo demasiado elevado, se ha optado por fijar un tiempo máximo de ejecución para cada uno de ellos de 30 minutos.

Respecto a los algoritmos de búsqueda empleados para la búsqueda de soluciones en los distintos tableros de juegos, se han utilizado principalmente los métodos de búsqueda en Amplitud, Profundidad, Profundidad Iterativa, A* e IDA*. En estos algoritmos se han realizado distintos tipos de mejoras para reducir los tiempos de búsqueda y poder así el éxito en la búsqueda de soluciones.

En los algoritmos de Amplitud y Profundidad, para evitar la generación de estados repetidos que pueden crear bucles (en especial en el caso de la búsqueda en Profundidad, que estos bucles lo hacen no completo) se creó una lista en la que se almacenan los estados generados. De este modo sólo se consideran los nuevos y se evita caer en ciclos.

Por otra parte se ha definido una función que encuentra un gran número de Deadlocks, evitándose de este modo expandir nodos que jamás llegarán a una solución. En concreto, esta función comprueba si hay piedras con muros en dos lados contiguos (en una esquina) o si se da el caso de que dos o más cajas estén contiguas en una pared.

Estas modificaciones consiguieron mejorar en una gran cantidad el éxito en la búsqueda de soluciones y se redujo en una magnitud elevada los tiempos para encontrar soluciones. Con las mejora de lista de nodos generados se pasó en algunos casos (Test 24) de no encontrar soluciones con amplitud y profundidad para tiempos de 10 minutos a encontrar soluciones en 64 y 15 segundos, respectivamente. Además, al emplearse la detección de Deadlocks se pasó a encontrar un mayor número de soluciones con todos los métodos (profundidad Iterativa en Test 24 no hallaba solución en 10 minutos, pasa a encontrar una en algo más de 5) y a reducir los tiempos de los que ya las encontraban, pasando a ser de media unas 6 veces menor (Test 24: A* pasa de 39 a 7 segundos e IDA* pasa de 182 a 30. Amplitud y Profundidad pasan a 18 y 2 segundos, respectivamente).

Para los algoritmos de Búsqueda Informada se ha considerado una Heurística que calcula la distancia de cada piedra a la base más próxima, las suma y añade la distancia del jugador a la caja más alejada. También se considera si el tablero se encuentra en situación de Deadlock, sumando en ese caso un valor de 1000000 o 0 en caso contrario. Esta Heurística es admisible ya que nunca sobrestima el coste para llegar a la meta desde un estado concreto:

- En el caso de las distancias a las bases, en el mejor de los casos los bloques acabarán en las bases a las que se les calculó la distancia, resultando el coste igual al esperado. En el resto de casos, acabarán alguna en otra base, que tendrá un coste igual o mayor al calculado, por lo que la heurística tendrá siempre un valor menor.
- Si consideramos la distancia del jugador a la piedra más alejada, para poder resolver el tablero, el jugador tiene que llevar todas las piedras a las bases, por lo que más tarde o más temprano el jugador deberá dirigirse a la piedra más alejada para moverla, por lo que esta parte también es admisible.
- Por último, la condición añadida para los Deadlock también es admisible, ya que si no hay Deadlock no añade ningún coste adicional al calculado, por lo que sigue siendo admisible, y si hay Deadlock la solución nunca será alcanzada (se puede considerar coste infinito) por lo que al añadir 1000000 sigue siendo menor que el coste real.

Se pretendía que la parte de búsqueda de soluciones fuera llamada desde la propia interfaz gráfica, pero el funcionamiento de la misma se entorpecía mucho al pasar demasiado tiempo sin llamar al bucle de juego mientras los algoritmos buscaban soluciones.

4.2 Análisis

4.2.1 Casos de Uso

En esta sección se detallarán los distintos casos de uso que tiene el software desarrollado. Para ello se incluirá un Diagrama de Casos de Uso, mostrando los casos de uso asociados a las cuatro acciones principales, y posteriormente se procederá a detallar los casos de uso.

Para el desarrollo de este diagrama se ha empleado el sitio web yUML [13], que permite diseñar distintos tipos de diagramas UML escribiendo las descripciones en una sintaxis concreta.

Ilustración 4: Diagrama de Casos de Uso

A continuación se procederá a explicar los casos de uso que aparecen en la *Ilustración 4*, siguiendo el formato concreto de plantilla en el que se cumple que:

- El identificador es único y sigue el formato de nombrado CU-XX donde XX es el número de caso de uso.
- Tiene también un nombre único.
- Se indicarán los actores, es decir el tipo de usuario que emplea el caso de uso.
- Se incluirán objetivos que se pretenden lograr con dicho caso.
- Se incluyen las precondiciones que deben ocurrir para poder ejecutar el caso de uso
- Las postcondiciones definen los hechos que ocurrirán tras desarrollarse el caso de uso.

4.2.1.1 Solucionar Niveles

En esta sección se recogen los casos de uso asociados a la actividad de solución de niveles.

Identificador	**CU-01**
Nombre	Buscar Soluciones
Actores	Usuario que desea encontrar soluciones para un conjunto de tableros de Sokoban.
Objetivos	Se buscan soluciones para los tableros mediante el empleo de distintas técnicas de Inteligencia Artificial.
Precondiciones	---
Postcondiciones	Se generan los ficheros: • NOMBREFICH_resultados.txt: se generan tantos como tableros, donde NOMBREFICH es el nombre del fichero que contiene el tablero. Recoge de forma comprensible los resultados de los algoritmos para ese tablero. • resumen.txt: fichero que contiene de forma resumida los resultados. • datos.txt: contiene todos los datos obtenidos en el proceso con un formato adecuado para ser leído por programas de análisis de datos.

Tabla 10: Caso de Uso CU-01

4.2.1.2 Jugar

Identificador	CU-02
Nombre	Seleccionar Jugar
Actores	El usuario que tras ejecutar la aplicación decide que desea jugar una partida de Sokoban.
Objetivos	El usuario selecciona y acepta la opción de “Jugar” en el menú.
Precondiciones	---
Postcondiciones	Se ofrece una lista de niveles posibles para jugar.

Tabla 11: Caso de Uso CU-02

Identificador	CU-03
Nombre	Escoger Nivel para Jugar.
Actores	La persona que quiere escoger el nivel para jugar.
Objetivos	El usuario selecciona el número de nivel deseado para jugar.
Precondiciones	El usuario debe haber seleccionado en la pantalla anterior la opción de jugar.
Postcondiciones	Se visualiza el tablero del nivel escogido.

Tabla 12: Caso de Uso CU-03

4.2.1.3 Mostrar Soluciones

Identificador	CU-04
Nombre	Seleccionar Mostrar Solución
Actores	La persona que está usando el sistema y decide visualizar la soluciones de niveles de Sokoban.
Objetivos	El usuario selecciona y acepta la opción de “Mostrar Solución” en el menú.
Precondiciones	---
Postcondiciones	Se ofrece una lista de niveles posibles para visualizar.

Tabla 13: Caso de Uso CU-04

Identificador	CU-05
Nombre	Escoger Nivel para Mostrar Solución
Actores	La persona que quiere escoger el nivel del cual mostrar una solución.
Objetivos	El usuario selecciona el número de nivel del que se mostrarán soluciones.
Precondiciones	El usuario debe haber seleccionado en la pantalla anterior la opción de “Mostrar Solución”.
Postcondiciones	Se visualiza mostrará la elección del algoritmo de búsqueda.

Tabla 14: Caso de Uso CU-05

Identificador	CU-06
Nombre	Escoger Algoritmo
Actores	Usuario que desea visualizar la solución hallada por un algoritmo concreto para el nivel escogido.
Objetivos	El usuario selecciona uno de los algoritmos de búsqueda disponibles en el menú.
Precondiciones	El usuario debe haber seleccionado en la pantalla anterior la el nivel a emplear.
Postcondiciones	Se visualiza el tablero del nivel escogido con la secuencia de movimientos para resolverlo.

Tabla 15: Caso de Uso CU-06

Identificador	CU-07
Nombre	Volver a Selección de Nivel
Actores	Usuario que desea elegir otro nivel distinto para visualizar su solución.
Objetivos	El usuario selecciona la opción de “Volver” en el menú de selección de Algoritmos.
Precondiciones	El usuario se encuentra en la pantalla de selección del Algoritmo para mostrar la solución.
Postcondiciones	Se muestra el menú de selección del nivel para mostrar.

Tabla 16: Caso de Uso CU-07

4.2.1.4 Salir

Identificador	CU-08
Nombre	Seleccionar Salir
Actores	Una persona que decide abandonar la interfaz gráfica del programa.
Objetivos	El usuario selecciona y acepta la opción de “Salir” en el menú o cierra la ventana del programa.
Precondiciones	---
Postcondiciones	Se finaliza el programa.

Tabla 17: Caso de Uso CU-08

4.2.2 Requisitos

A continuación se incluyen los requisitos que debe cumplir el sistema. Se distinguirá entre requisitos funcionales y no funcionales.

4.2.2.1 Requisitos Funcionales

Los requisitos funcionales son los que hacen referencia a las funcionalidades del sistema, detallan lo que debe realizar el sistema.

A continuación se recogen estos requisitos, representados en una tabla. El indicador de estos requisitos es *RF_XX*, donde XX representa el número de requisito al que referencia.

Id	**RF_01**		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Se podrán leer niveles en notación XSB desde ficheros de texto txt.		

Tabla 18: Requisito Funcional RF_01

Id	RF_02		
Prioridad	☐ Alta ☐ Media ☑ Baja	Necesidad	☐ Esencial ☐ Deseable ☑ Opcional
Fuente	☐ Cliente	☑ Desarrollador.	
Descripción	Durante la ejecución del solucionador se mostrará el progreso (tablero ejecutando) y resultado (ÉXITO o FRACASO) de los niveles.		

Tabla 19: Requisito Funcional RF_02

Id	RF_03		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Se dará la posibilidad a los usuarios de jugar al juego de Sokoban con el programa.		

Tabla 20: Requisito Funcional RF_03

Id	RF_04		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Los usuarios del programa tendrán la posibilidad de visualizar soluciones generadas.		

Tabla 21: Requisito Funcional RF_04

Id	RF_05		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Se podrá salir del programa en cualquier momento.		

Tabla 22: Requisito Funcional RF_05

Id	RF_06		
Prioridad	☐ Alta ☐ Media ☑ Baja	Necesidad	☐Esencial ☐ Deseable ☑ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	En los menús habrá una opción para salir del programa.		

Tabla 23: Requisito Funcional RF_06

Id	RF_07		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Los usuarios podrán escoger el nivel sobre el que jugar.		

Tabla 24: Requisito Funcional RF_07

Id	RF_08		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Se podrá seleccionar el nivel para mostrar sus soluciones.		

Tabla 25: Requisito Funcional RF_08

Id	RF_09		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Se podrá seleccionar la solución a mostrar de las encontradas por los algoritmos.		

Tabla 26: Requisito Funcional RF_09

Id	RF_10		
Prioridad	☐ Alta ☑ Media ☐ Baja	Necesidad	☐ Esencial ☑ Deseable ☐ Opcional
Fuente	☑ Cliente		☐ Desarrollador.
Descripción	Se podrá volver al menú de selección de niveles desde el menú de selección de algoritmos.		

Tabla 27: Requisito Funcional RF_10

Id	RF_11		
Prioridad	☐Alta ☑ Media ☐ Baja	Necesidad	☐ Esencial ☑ Deseable ☐ Opcional
Fuente	☑ Cliente		☐ Desarrollador.
Descripción	Se permitirá jugar a las partidas cargadas con las teclas de dirección del teclado.		

Tabla 28: Requisito Funcional RF_11

4.2.2.2 Requisitos No Funcionales

Los requisitos no funcionales son los que describen restricciones que debe seguir el sistema, relaciones de entrada salida, tipos de datos, fiabilidad, etc

A continuación se recogen estos requisitos, representados en una tabla. El indicador de estos requisitos es *RNF_XX*, donde XX representa el número de requisito al que referencia.

Id	**RNF_01**		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Las soluciones encontradas se representarán con notación LURD.		

Tabla 29: Requisito No Funcional RNF_01

Id	RNF_02		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☐ Cliente	☑ Desarrollador.	
Descripción	Si no se encuentra una solución, se fijará el valor a “None”.		

Tabla 30: Requisito No Funcional RNF_02

Id	RNF_03		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Se podrá definir el tiempo máximo de ejecución de los algoritmos.		

Tabla 31: Requisito No Funcional RNF_03

Id	RNF_04		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☐ Esencial ☑ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	El tiempo de respuesta de la interfaz no será superior a 1 segundo.		

Tabla 32: Requisito No Funcional RNF_04

Id	RNF_05		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☐ Cliente	☑ Desarrollador.	
Descripción	Para poder visualizar las soluciones de los tableros debe haberse ejecutado la búsqueda de soluciones primero.		

Tabla 33: Requisito No Funcional RNF_05

Id	RNF_06		
Prioridad	☐ Alta ☑ Media ☐ Baja	Necesidad	☐ Esencial ☑ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Para cada nivel a solucionar se generará un txt detallado con los resultados de los distintos algoritmos.		

Tabla 34: Requisito No Funcional RNF_06

Id	RNF_07		
Prioridad	☐ Alta ☑ Media ☐ Baja	Necesidad	☐ Esencial ☑ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Se generará un fichero con el resumen de soluciones encontradas para cada nivel y de número de niveles resueltos con cada método.		

Tabla 35: Requisito No Funcional RNF_07

Id	RNF_08		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Se generará un fichero de datos separados con comas con una línea por tablero de Sokoban analizado para poder ser analizado con herramientas de análisis de datos.		

Tabla 36: Requisito No Funcional RNF_08

Id	RNF_09		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☑ Cliente	☐ Desarrollador.	
Descripción	Los algoritmos de búsqueda almacenarán el número de cajas movidas, pasos del jugador, tiempo de búsqueda, nodos generados y nodos expandidos.		

Tabla 37: Requisito No Funcional RNF_09

Id	RNF_10		
Prioridad	☑ Alta ☐ Media ☐ Baja	Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Fuente	☐ Cliente	☑ Desarrollador.	
Descripción	Si no hay solución para un nivel con un determinado algoritmo, se mostrará un mensaje de error y se volverá al menú de selección de algoritmo.		

Tabla 38: Requisito No Funcional RNF_10

4.3 Diseño

En este apartado se describirá la estructura del diseño realizado para el proyecto. Se definirán las clases que componen el sistema y sus relaciones, incluyendo un diagrama de clases para una mayor comprensión.

4.3.1 Definición de las Clases

Par realizar el diagrama de clases siguiente, cada una de las clases ha sido desarrollada por separado con la ayuda de la web yUML [13], aunque las relaciones han sido realizadas con programas de edición de imágenes, para que la estructura estuviera más organizada.

Cabe aclarar que “Conjunto de Reglas” no es una clase como tal, sino un fichero que contiene varias funciones que emplean distintas clases.

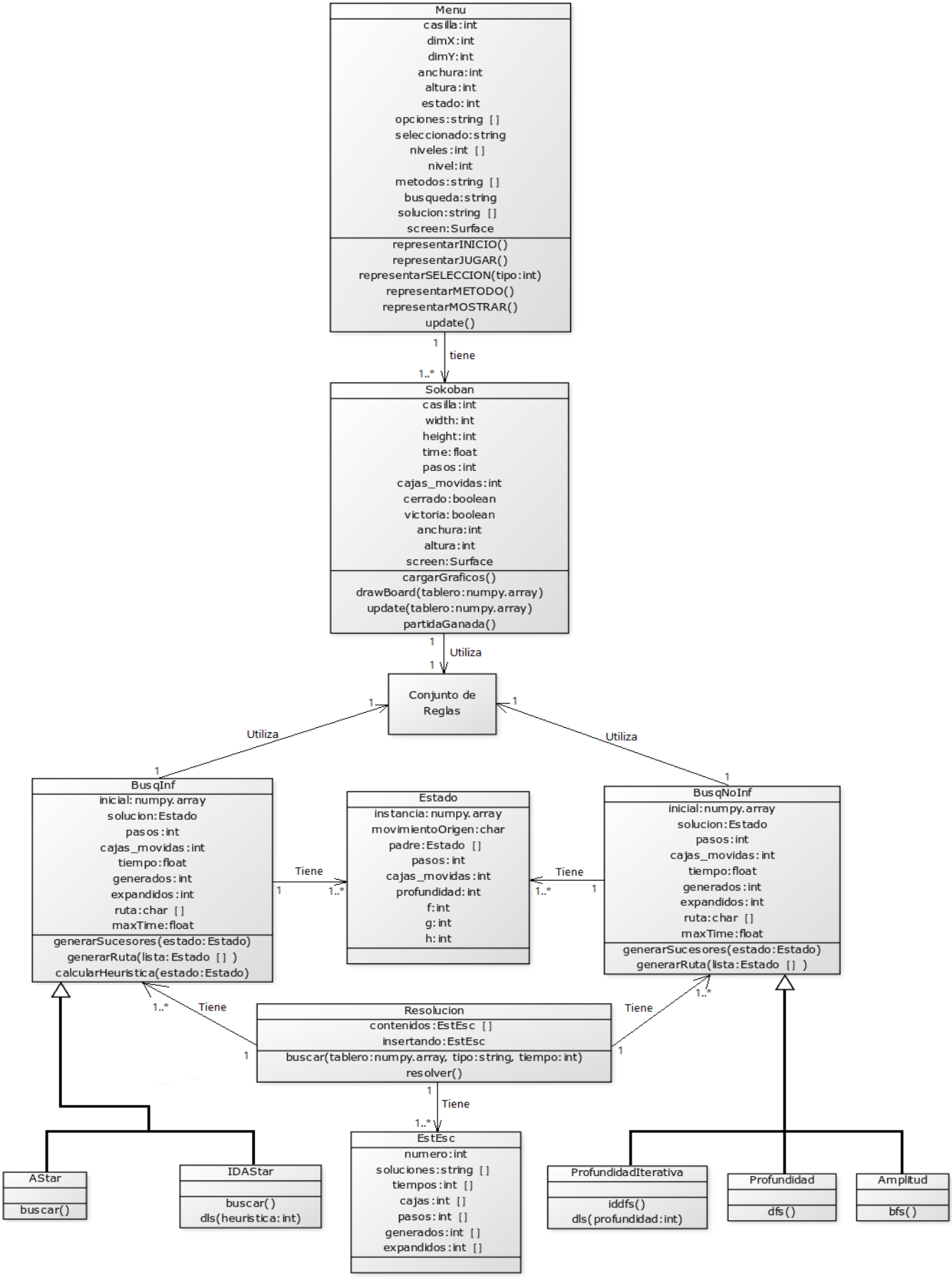

Ilustración 5: Diagrama de Clases

A continuación se procederá a describir en detalle cada una de las clases. Se comentarán también las funciones que se incluyen en “Conjunto de Reglas”.

Nombre	Sokoban
Responsabilidades	Genera la representación gráfica y mecánica que permite jugar con un nivel de juego.
Atributos	• **casilla:** atributo de tipo entero que refleja el tamaño del lado de la casilla en pixeles. • **width:** atributo entero que representa la longitud del tablero en píxeles. • **height:** altura en pixeles del nivel de Sokoban • **time:**variable auxiliar, mide el tiempo al comienzo de la partida en segundos y se utiliza para obtener el tiempo de ejecución al restar los tiempos medidos en distintos intervalos. Es un decimal. • **pasos**: entero que refleja los pasos del jugador. • **cajas_movidas:** entero para representar el número de empujones de cajas. • **cerrado:** variable booleana para saber si se ha cerrado la ventana y saber cuándo parar el bucle de juego. • **victoria:** estado booleano para saber si se ha Ganado la partida o no. • **anchura:** ancho de la ventana de juego en píxeles • **altura:** entero que recoge el alto de ventana en píxeles • **screen:** objeto de tipo Surface de Pygame que representa la salida a dibujar por pantalla.
Operaciones	• **cargarGraficos():** asigna los ficheros de imágenes a las variables que los emplearán. • **drawBoard(tablero: numpy.array):** recibe un tablero de Sokoban y la dibuja en pantalla. • **update(tablero: numpy.array):** es la función principal del juego que se ejecuta en un bucle para actualizar en cada instante la pantalla. Actualiza el rompecabezas según los eventos ocurridos (teclas) y llama a *drawboard()*. Comprueba si se ha completado la partida o se ha cerrado la ventana. • **partidaGanada():** función que es llamada cuando el tablero está resuelto. Muestra un mensaje indicando la victoria.

Tabla 39: Clase Sokoban

Nombre	Menu
Responsabilidades	Es la clase encargada de generar el menú de la Interfaz Gráfica.
Atributos	• **casilla:** atributo de tipo entero que refleja el tamaño del lado de la casilla en pixeles. • **dimX:** atributo entero que representa la longitud de la ventana en casillas. • **dimY:** altura en casillas de la ventana. • **anchura:** ancho de la ventana de juego en píxeles • **altura:** el alto de ventana en píxeles. • **estado:** entero que representa el menú actual de la interfaz. • **opciones:** array de strings con las opciones del primer menú. • **seleccionado**: opción que se muestra resaltada en el primer menú. • **niveles:** array de enteros que representa los distintos niveles en el menú de selección de nivel. • **nivel:** entero que indica el valor resaltado en la selección de niveles. • **metodos:** array de strings con los algoritmos de búsqueda del menú de selección de algoritmo. • **busqueda:** string del algoritmo seleccionado. • **solucion:** array de strings con las soluciones de los distintos métodos para el nivel seleccionado. • **screen:** objeto de tipo Surface de *Pygame* que representa la salida a dibujar por pantalla.
Operaciones	• **representarINICIO():** carga el menú de inicio. • **representarJUGAR():** utiliza la clase *Sokoban* para gestionar el modo de juego a una partida. • **representarSELECCION(tipo:int):** muestra la pantalla de selección de nivel. El parámetro indica si es nivel para jugar o para resolver. • **representarMETODO():** gestiona el menú de selección del algoritmo de búsqueda. • **representarMOSTRAR():** función encargada de emplear la clase *Sokoban* y pasarle los eventos de tecla según las solución del algoritmo. • **update():** es la función principal del menú que se ejecuta en un bucle para actualizar en cada instante la pantalla. Comprueba el estado actual para llamar a la función correspondiente.

Tabla 40: Clase Menú

Nombre	BusqNoInf
Responsabilidades	Clase de la que heredan los algoritmos de búsqueda no informada. Contiene los atributos y funciones comunes a todos ellos.
Atributos	• **inicial:** tablero de tipo numpy.array que representa el estado inicial de juego sobre el que se realizará la búsqueda. • **solucion:** instancia de la clase *Estado* que representa el estado solución. • **pasos**: entero que refleja los pasos del jugador del al resolverse el juego. • **cajas_movidas:** entero para representar el número de empujones de cajas de la solución. • **tiempo:** variable auxiliar, mide el tiempo al comienzo de la partida en segundos y se utiliza para obtener el tiempo de ejecución al restar los tiempos medidos en distintos intervalos. Es un número decimal. • **generados:** número de nodos generados durante la búsqueda. • **expandidos:** entero que representa el número de nodos expandidos durante el algoritmo. • **ruta:** array de caracteres donde cada uno de ellos representa un movimiento de la solución. • **maxTime:** el máximo tiempo que ejecutará el algoritmo.
Operaciones	• **generarSucesores(estado: Estado):** función que recibe un estado y con ayuda de las funciones de las reglas de juego calcula los estados alcanzables en un paso desde ese estado. • **generarRuta(lista:Estado[]):** recibe un array de objetos *Estado* y genera el array de caracteres de la solución que genera esa ruta.

Tabla 41: Clase BusqNoInf

Nombre	Amplitud
Responsabilidades	Clase que representa un algoritmo de Búsqueda en Amplitud. Hereda de la clase *BusqNoInf*.
Atributos	• Los mismos que la clase *BusqNoInf.*
Operaciones	• Las mismas que la clase *BusqNoInf.* • **bfs():** ejecuta el algoritmo de Búsqueda en Amplitud.

Tabla 42: Clase Amplitud

Nombre	Profundidad
Responsabilidades	Clase que representa un algoritmo de Búsqueda en Profundidad. Hereda de la clase *BusqNoInf*.
Atributos	• Los mismos que la clase *BusqNoInf.*
Operaciones	• Las mismas que la clase *BusqNoInf.* • **dfs():** ejecuta el algoritmo de Búsqueda en Profundidad.

Tabla 43: Clase Profundidad

Nombre	ProfundidadIterativa
Responsabilidades	Clase que representa un algoritmo de Búsqueda en Profundidad Iterativa. Hereda de la clase *BusqNoInf*.
Atributos	• Los mismos que la clase *BusqNoInf.*
Operaciones	• Las mismas que la clase *BusqNoInf.* • **iddfs():** ejecuta el algoritmo de Búsqueda en ProfundidadIterativa. • **dls(profundidad:int):** ejecuta una búsqueda en profundidad limitada.

Tabla 44: Clase ProfundidadIterativa

Nombre	BusqInf
Responsabilidades	Clase de la que heredan los algoritmos de búsqueda informada. Contiene los atributos y funciones comunes a todos ellos.
Atributos	• **inicial:** tablero de tipo numpy.array que representa el estado inicial de juego sobre el que se realizará la búsqueda. • **solucion:** instancia de la clase *Estado* que representa el estado solución. • **pasos**: entero que refleja los pasos del jugador del al resolverse el juego. • **cajas_movidas:** entero para representar el número de empujones de cajas de la solución. • **tiempo:** variable auxiliar, mide el tiempo al comienzo de la partida en segundos y se utiliza para obtener el tiempo de ejecución al restar los tiempos medidos en distintos intervalos. Es un número decimal. • **generados:** número de nodos generados durante la búsqueda. • **expandidos:** entero que representa el número de nodos expandidos durante el algoritmo. • **ruta:** array de caracteres donde cada uno de ellos representa un movimiento de la solución. • **maxTime:** el máximo tiempo que ejecutará el algoritmo.
Operaciones	• **generarSucesores(estado: Estado):** función que recibe un estado y con ayuda de las funciones de las reglas de juego calcula los estados alcanzables en un paso desde ese estado. • **generarRuta(lista:Estado[]):** recibe un array de objetos *Estado* y genera el array de caracteres de la solución que genera esa ruta. • **calcularHeuristica(estado:Estado):** calcula la heurística del estado recibido.

Tabla 45: Clase BusqInf

Nombre	AStar
Responsabilidades	Clase que representa un algoritmo de Búsqueda en A*. Hereda de la clase *BusqInf*.
Atributos	• Los mismos que la clase *BusqInf.*
Operaciones	• Las mismas que la clase *BusqInf.* • **buscar():** ejecuta el algoritmo de Búsqueda en A*.

Tabla 46: Clase AStar

Nombre	IDAStar
Responsabilidades	Clase que representa un algoritmo de Búsqueda en IDA*. Hereda de la clase *BusqInf*.
Atributos	• Los mismos que la clase *BusqInf.*
Operaciones	• Las mismas que la clase *BusqInf.* • **buscar():** ejecuta el algoritmo de Búsqueda en IDA*. • **dls(profundidad:int):** ejecuta una búsqueda en profundidad limitada.

Tabla 47: Clase IDAStar

Nombre	Resolucion
Responsabilidades	Clase que invoca los algoritmos de Búsqueda sobre el conjunto de niveles.
Atributos	• **contenidos:** array de EstEsc que almacena las distintas líneas de datos para escribir en el fichero datos.txt. • **insertando:** EstEsc en el que se están almacenando los resultados de las búsquedas para el nivel actual.
Operaciones	• **buscar(tablero: numpy.array, tipo: string, tiempo: int):** invoca el algoritmo *tipo* sobre *tablero* con tiempo máximo *tiempo*. • **resolver():** para cada instancia en la carpeta "levels" invoca *buscar()* con cada algoritmo.

Tabla 48: Clase Resolucion

Nombre	EstEsc
Responsabilidades	Clase utilizada para almacenar la información relativa a los datos de ejecución de los algoritmos sobre un nivel.
Atributos	• **numero:** número del nivel al que corresponde la información. • **soluciones:** array de strings con las correspondientes soluciones halladas para ese nivel. • **tiempos:** array con los tiempos de los algoritmos. • **cajas:** array de int para almacenar las cajas movidas por los distintos algoritmos. • **pasos:** array del número de pasos dados por el jugador hasta llegar a ese estado desde el inicial. Una posición por cada uno de los algoritmos. • **generados:** array del número de nodos generados por los algoritmos. • **expandidos:** array de nodos expandidos por los algoritmos.
Operaciones	• No tiene.

Tabla 49: Clase EstEsc

Nombre	Estado
Responsabilidades	Clase utilizada para almacenar la información relativa a los nodos de los algoritmos de búsqueda.
Atributos	• **instancia:** atributo numpy.array que almacena las casillas del nodo. • **movimientoOrigen:** carácter que representa el movimiento con el que se ha llegado hasta ese nodo. • **padre:** array de estados que contiene los antecesores de ese estado. • **pasos:** número de pasos dados por el jugador hasta llegar a ese estado desde el inicial. • **cajas_movidas:** int con el total decajas empujadas hasta llegar a ese estado. • **profundidad:** Profundidad a la que se encuentra ese nodo. Sólo se emplea en la Búsqueda en Profundidad Iterativa. • **f:** valor entero, toma como valor la suma de g+h. Se usa sólo en Búsquedas Informadas. • **g:** valor entero que representa el coste real desde el inicio hasta ese estado. Se usa sólo en Búsquedas Informadas. • **h:** heurística del coste que se estima que hay desde ese estado hasta el nodo meta. Se usa sólo en Búsquedas Informadas.
Operaciones	• No tiene.

Tabla 50: Clase Estado

Nombre	Conjunto de Reglas
Responsabilidades	Conjunto de funciones utilizadas por varias clases.
Atributos	• No tiene
Operaciones	• **buscarJugador(tablero:numpy.array):** devuelve la posición del jugador en el tablero. • **bloqueo(tablero:numpy.array**): comprueba si hay Deathlock, devolviendo True o False. • **cajaBloqueada(tablero:numpy.array, posicion:int[]):** devuelve un booleao que indica si en esa posición hay una cajabloqueada. • **posicionDesplazada(inicial:int[], tecla:char):** devuelve la posición resultante de desplazarse en la dirección indicada por la tecla. • **puedeMover(tablero:numpy.array, destinoJugador:int[], destinoCaja:int[]):** comprueba si es posible realizar un movimiento. • **realizarMovimiento(tablero:numpy.array, tecla:char, pasos:int, cajas:int):** intenta realizar un movimiento, devolviendo el valor booleano de la operación y actualizando los pasos y cajas según convenga. • **resuelto(tablero:numpy.array):** comprueba si el tablero está resuelto.

Tabla 51: Funciones del Conjunto de Reglas

4.3.2 Clases Asociadas a los Casos de Uso

En esta sección se relacionarán los casos de uso definidos en el capítulo *Casos de Uso* con las clases identificadas en el apartado *Definición de las Clases*. Las funciones de "Conjunto de Reglas" son utilizadas en todos los Casos de Uso excepto CU-08

Caso de Uso	Clases que intervienen
CU-01	• Resolucion • EstEsc • BusqInf • AStar • IDAstar • BusqNoInf • Amplitud • Profundidad • ProfundidadIterativa • Estado
CU-02	• Menu • Sokoban
CU-03	• Menu • Sokoban
CU-04	• Menu • Sokoban
CU-05	• Menu • Sokoban
CU-06	• Menu • Sokoban
CU-07	• Menu • Sokoban
CU-08	• Menu • Sokoban

Tabla 52: Relación entre Casos de Uso y Clases

5 Pruebas y Resultados

A continuación de procederá a analizar los resultados obtenidos con los distintos algoritmos y se compararán las soluciones según las siguientes características:

- Solución encontrada o no
- Optimalidad
- Tiempo de ejecución
- Nodos Expandidos y Generados
- Uso de Memoria

En el *ANEXO III: Resultados* se encuentran todos los datos obtenidos de las distintas ejecuciones de los algoritmos. En este apartado se procederá a analizar dichos resultados.

5.1 Solución Encontrada y Optimalidad

En primer lugar se analizará el número de problemas que se ha logrado resolver con cada una de las técnicas empleadas así como el número de problemas para los que se ha alcanzado una solución con alguna de los algoritmos del estudio.

	Problemas Resueltos
Amplitud	51/61
Profundidad	53/61
Profundidad Iterativa	35/61
A*	52/61
IDA*	41/61
Total Resueltos	55/61

Tabla 53: Número de Problemas Resueltos

Así, como puede observarse en la Tabla 2, hay un total de 6 problemas para los que no se ha hallado ninguna solución. Estos conjuntos de Test son las instancias 51, 52, 55, 57, 58 y 61. La lista completa de los problemas con su representación pueda encontrarse en el *ANEXO II: 61 Kids Problems*.

Por otra parte, puede verse que el método que más soluciones ha encontrado es la búsqueda en profundidad. Este algoritmo apenas encontraba soluciones antes de aplicar la mejora mencionada en el *Decisiones Tomadas*, consiguiéndose una gran mejora en la probabilidad de encontrar soluciones. Sin embargo, aunque este método encuentra un gran número de soluciones, en algunos casos en tiempos muy bajos, no puede asegurarse que estas soluciones sean óptimas, por lo que en caso de haberse encontrado la solución con otro método que asegura optimalidad, se preferirá dicha solución. De los problemas resueltos por el Algoritmo de Búsqueda en Profundidad, hay un total de 3 problemas que no has sido resueltos por ningún otro algoritmo: 1, 50 y 59.

5.2 Tiempo de Ejecución y Mejor Solución

En vista de los resultados anteriores, se han analizado con Excel los resultados, creando una función que busca para cada problema el algoritmo que encuentra solución en un menor tiempo, de entre los que aseguran que se encuentra una solución óptima, y en caso de no encontrarse solución entre estos algoritmos, el mejor será el de profundidad, si ha encontrado una solución. La siguiente tabla refleja los resultados de este análisis:

	Amplitud	Profundidad	Profundidad Iterativa	A*	IDA*	Mejor Solución
1	None	112.3095299	None	None	None	Profundidad
2	0.863171559	0.436729747	28.74760294	0.640660255	6.383325092	A*
3	18.79831147	3.53422438	9.408098346	0.809477594	0.501524177	IDA*
4	2.138465127	1.176407786	5.253758119	0.393853776	0.543395429	A*
5	17.04847476	8.048066447	626.8593114	4.008276446	11.98368991	A*
6	0.557234418	0.550037241	4.441087175	0.526635985	1.337081364	A*
7	3.418728939	1.229099084	15.57166844	1.374596557	3.155372987	A*
8	285.7781213	136.8900028	None	35.97795843	605.6267748	A*
9	2.568935388	16.99686273	3.900334519	0.347841907	0.561499513	A*
10	34.643381	3.400090007	667.1194829	15.7543747	105.4315372	A*
11	11.99543876	16.68756399	51.07917305	2.597830793	4.499110527	A*
12	10.74599498	18.26070702	65.93348544	2.875217923	14.36945464	A*
13	1.507203675	0.190395876	9.252064666	1.121910867	3.092724784	A*
14	1.142136556	0.553274363	32.87167097	1.195597295	19.17239096	Amplitud
15	25.53950377	8.196494633	158.4099635	6.751073284	19.33427923	A*
16	60.81368455	63.67956523	None	44.98288012	None	A*
17	6.780900225	2.709171254	171.9417373	4.166550054	64.49574551	A*
18	None	None	None	319.8914213	519.9634704	A*
19	202.0855481	32.21772504	None	21.59357459	81.33361922	A*
20	1769.723384	1782.276134	None	1139.248365	None	A*
21	26.23239092	11.34940609	272.3941171	12.00706928	27.65959944	A*
22	8.481752629	13.75600865	None	7.697919638	943.6302855	A*
23	22.84414825	9.52830892	299.4279711	6.915928517	29.27268362	A*
24	18.36280736	2.822628533	351.1528581	7.548715171	30.64599641	A*

25	0.087517894	0.141715244	0.289619527	0.074399785	0.134340924	A*
26	12.74842709	7.353760787	1866.761804	10.30354538	285.1024382	A*
27	0.288918481	0.451102899	0.452170541	0.160910224	0.162879309	A*
28	39.48657924	13.52566957	None	14.49463956	112.39294	A*
29	37.37556776	6.083038039	None	33.26984182	None	A*
30	55.95081462	6.493351007	231.0439224	15.14652271	39.70678154	A*
31	69.24646929	4.866910699	1864.931688	54.7457939	487.9582402	A*
32	1.45234016	0.130541597	2.485179204	0.492121269	0.512967982	A*
33	33.32259468	5.597717258	1000.425856	4.590908562	23.71428171	A*
34	38.41261582	37.80939005	None	20.71787038	142.0081868	A*
35	35.97178923	6.070434259	None	33.11695704	None	A*
36	5.866786221	2.110960871	72.82199657	3.162957963	10.69986205	A*
37	59.7954642	50.80631434	None	47.45991966	None	A*
38	25.31643163	7.519258929	1605.147556	16.23351533	123.7311232	A*
39	10.73624941	19.10785985	1011.33406	4.679485207	334.8708578	A*
40	24.04385949	22.3139359	975.7005398	15.25058324	78.93416734	A*
41	4.3921911	2.215441341	410.8481562	4.223478405	92.27269867	A*
42	8.694651737	2.6122101	266.1361438	6.708516717	38.97624514	A*
43	130.5211405	40.99995144	None	54.49253643	None	A*
44	30.37289353	18.1235284	1000.273899	12.13185546	57.25049364	A*
45	16.08633672	5.617620807	877.6156531	10.82692372	161.1661383	A*
46	9.963871169	4.446921246	None	8.723971189	None	A*
47	22.54716945	11.78891882	70.40455687	2.23138552	5.586345266	A*
48	11.28164128	1.210583263	24.78537461	2.335119804	4.946275257	A*
49	850.9594775	None	None	134.9599802	554.6639487	A*
50	None	11.26938768	None	None	None	Profundidad
51	None	None	None	None	None	Ninguna
52	None	None	None	None	None	Ninguna
53	5.132955931	1.672824928	487.4749043	4.643971928	162.2350645	A*
54	285.8669818	214.3458988	None	228.2850337	None	A*
55	None	None	None	None	None	Ninguna
56	748.2329011	626.9476137	None	319.7537831	None	A*
57	None	None	None	None	None	Ninguna
58	None	None	None	None	None	Ninguna
59	None	89.74548749	None	None	None	Profundidad
60	262.680704	88.08072493	None	190.13338	None	A*
61	None	None	None	None	None	Ninguna

Tabla 54: Mejor solución para cada nivel en función del tiempo

De los resultados obtenidos en esta tabla, si tomamos en cuenta el tiempo empleado para encontrar una solución y la optimalidad de la misma, el algoritmo que mejores resultados ofrece es el A* en la gran mayoría de los supuestos, dándose únicamente un caso que es mejor con el algoritmo de Amplitud y otro que tiene un mejor tiempo con la búsqueda en IDA*.

Con los datos de tiempos totales se ha creado también una representación gráfica:

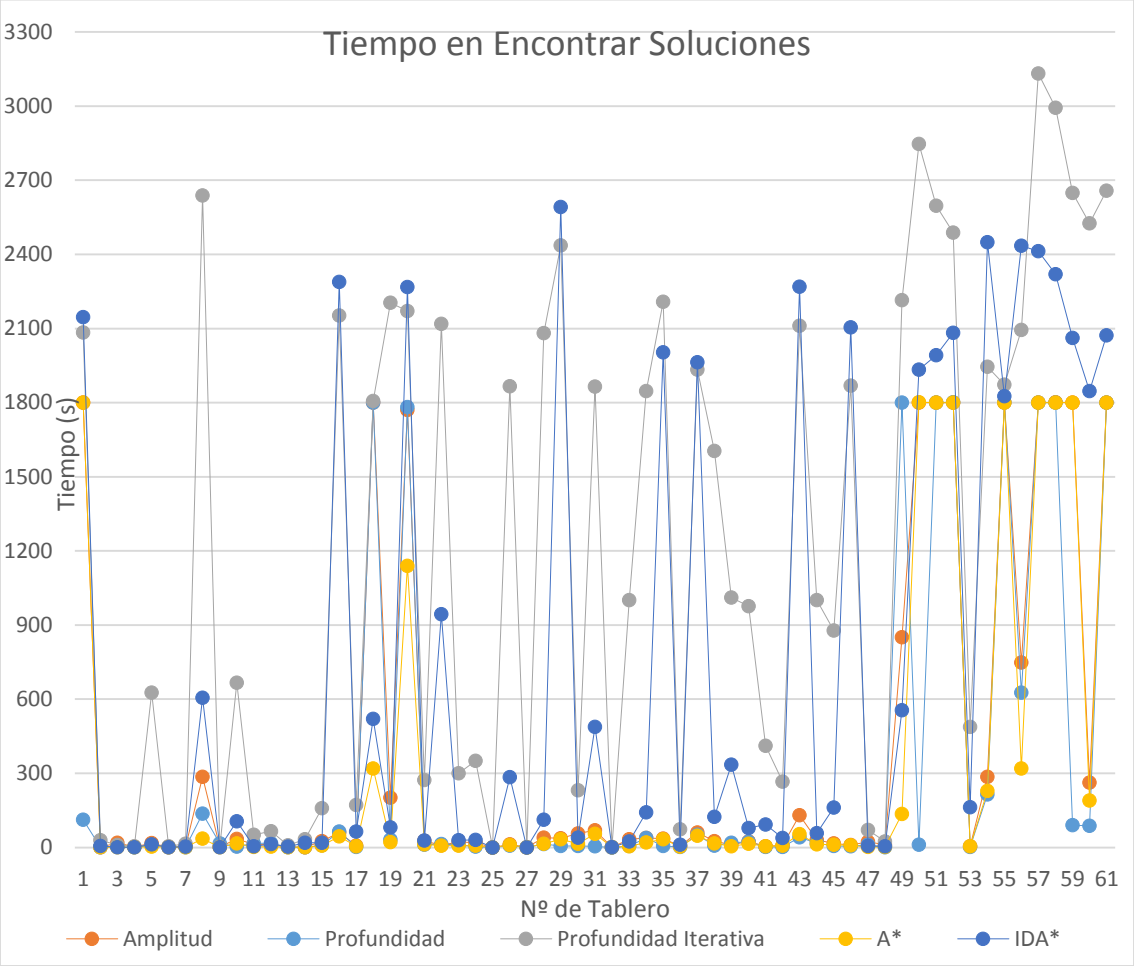

Ilustración 6: Mejor solución para cada nivel en función del tiempo

Los tiempos que sobrepasan los 1800 segundos son algoritmos que no han encontrado solución y han terminado al agotarse el tiempo máximo. En los algoritmos iterativos el tiempo de ejecución supera el tiempo máximo con una diferencia muy elevada debido a que la comprobación de tiempo se realiza en la función principal del algoritmo, no en la profundidad limitada, por lo que pasa más tiempo hasta que se comprueba la condición de tiempo.

Tal y como habíamos comentado antes, los algoritmos más rápidos en este dominio son Profundidad y A*, más o menos balanceados estando los casos en los que cada uno es el más rápido. Están seguidos de muy cerca por Amplitud.

Puede observarse además que los últimos niveles son los que presentan tiempos más elevados, en la gran mayoría de casos superando el límite y no encontrando por tanto una solución. Un caso especial es el primer problema, donde únicamente Profundidad es capaz de hallar una solución.

5.3 Nodos Generados

En la siguiente tabla y gráfico se recoge la cantidad de nodos generados y expandidos. En el caso del gráfico, el eje Y tiene un escalado logarítmico para apreciar mejor los datos, ya que la diferencia entre magnitudes es muy grande en algunos casos.

Nivel	Amplitud	Profundidad	Profundidad Iterativa	A*	IDA*
1	5529	649	252193	7461	256944
2	75	52	4880	88	1387
3	564	162	2658	125	205
4	163	105	1403	79	222
5	483	275	88482	297	2650
6	58	58	716	68	285
7	197	86	2612	130	794
8	2009	1173	351075	1013	87794
9	190	353	1066	67	197
10	715	125	77321	609	15661
11	432	348	10504	236	1311
12	397	365	11111	240	2902
13	125	32	1964	137	923
14	96	55	4387	119	3051
15	627	223	27886	400	4342
16	882	767	206853	950	210252
17	303	128	24146	281	10635
18	5490	4120	260371	3280	84447
19	1779	621	210943	737	10163
20	4965	4195	242303	5935	241381
21	603	364	34011	551	5014
22	328	334	233009	397	113875
23	581	277	43458	412	5846
24	514	143	46728	425	5900
25	20	25	104	23	70
26	397	245	186003	480	36531
27	52	52	183	44	82
28	766	381	251566	615	19723
29	675	210	216795	863	240575
30	968	239	29556	600	6862
31	1015	201	166848	1151	53826
32	139	32	815	96	246
33	716	204	126786	327	5015
34	733	657	189298	701	20720
35	688	196	169464	783	155242
36	265	120	9193	243	2037
37	916	810	218386	1032	213678

38	580	256	160038	581	16911
39	327	383	65810	248	28066
40	573	495	110149	608	13005
41	213	127	40201	263	11874
42	304	133	25860	342	5544
43	1340	614	178680	1136	183913
44	651	445	100140	514	8365
45	449	238	73174	471	18807
46	325	178	134771	356	151069
47	606	349	10410	208	1282
48	399	94	3790	208	1074
49	3758	4551	268084	2070	79160
50	6616	283	420458	7969	250885
51	5473	4221	259786	7149	203526
52	6449	2989	285281	7928	221328
53	234	120	43415	263	16399
54	2004	1472	106128	2210	122343
55	5408	3591	261511	7214	249573
56	3422	2997	177544	2954	199021
57	6074	4747	418998	7643	291121
58	6117	4821	398207	7590	278805
59	5987	977	286524	7436	237634
60	2040	1083	288985	2242	206392
61	5925	4341	304369	7328	213535

Tabla 55: Nodos Generados

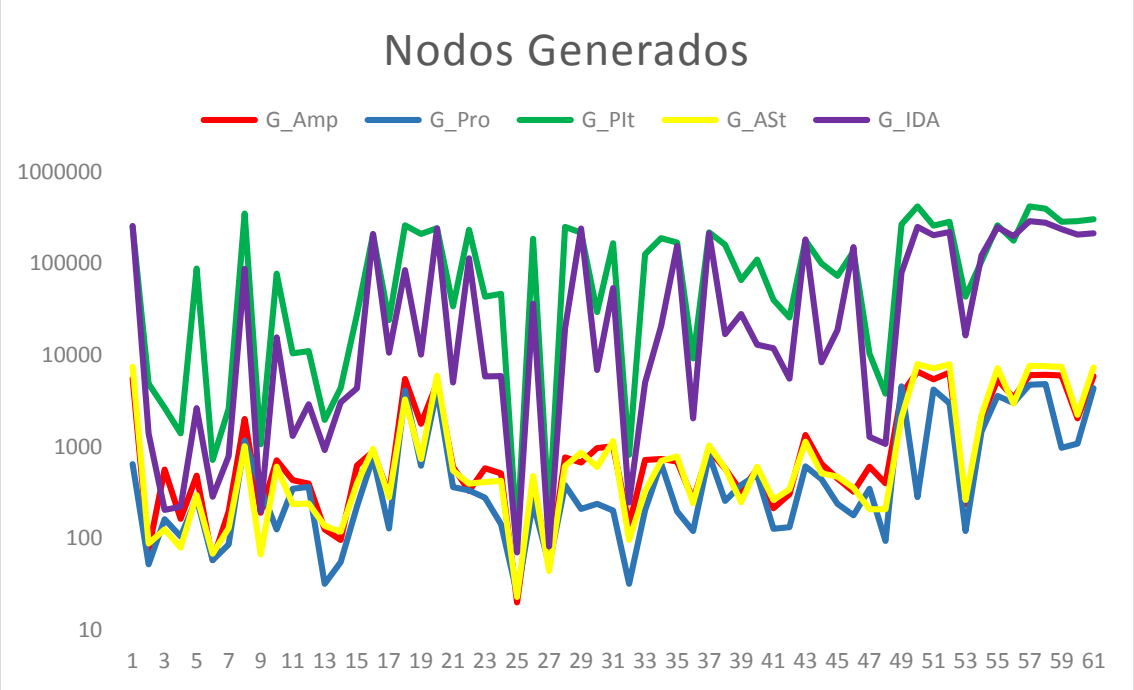

Ilustración 7: Nodos Generados

En este caso se distingue con bastante claridad que el algoritmo que menos nodos genera es el de búsqueda en profundidad. Esto es debido en una gran parte a la mejora de la lista de nodos generados que evita que se generen nodos repetidos, evitado así los bucles. Tras este algoritmo se encuentra A*, con una diferencia mínima respecto al algoritmo de Primero en Amplitud. Profundidad Iterativa e IDA* son los que más nodos generan, lo cual es comprensible si se tiene en cuenta que en cada nueva iteración vuelve a generar el árbol.

5.4 Nodos Expandidos

Al igual que en el caso anterior, ahora se muestran los nodos expandidos por los métodos de búsqueda.

Nivel	Amplitud	Profundidad	Profundidad Iterativa	A*	IDA*
1	4873	550	154169	4492	133111
2	73	43	3588	54	871
3	470	153	1567	65	100
4	144	95	911	43	121
5	445	258	56920	164	1459
6	53	52	556	45	183
7	175	79	2139	77	468
8	1890	1101	213229	537	48982
9	153	333	659	39	107
10	661	95	53731	400	9700
11	375	339	6902	146	737
12	349	352	7315	148	1720
13	118	26	1580	89	553
14	91	47	3914	87	2286
15	571	202	18292	256	2524
16	852	724	157080	666	148091
17	282	116	18739	203	7369
18	5028	3961	162355	1912	44747
19	1673	602	132988	425	5778
20	4922	4087	156469	3775	141144
21	577	347	24871	340	3016
22	320	322	177043	286	79173
23	547	248	29857	254	3384
24	484	117	33373	266	3503
25	18	24	92	14	39
26	389	219	138680	323	23440
27	44	49	130	23	39
28	704	360	173015	369	11593
29	667	174	151466	584	142469

30	892	220	21985	413	4409
31	996	183	126666	835	35886
32	120	22	553	52	118
33	643	166	87049	183	2879
34	715	637	138290	480	13012
35	678	163	133586	620	108557
36	250	94	6881	152	1270
37	897	793	166802	765	152664
38	570	231	115529	404	10835
39	297	369	56290	177	20877
40	559	482	80144	400	7941
41	209	108	32098	187	8028
42	299	111	20364	235	3546
43	1302	580	135443	782	132908
44	631	408	71460	352	5375
45	428	205	59268	317	12795
46	312	157	116485	272	113671
47	495	312	7101	116	709
48	339	77	2750	118	628
49	3442	4470	167031	1205	43591
50	4689	214	239213	4479	133676
51	5002	4092	181696	4696	132412
52	4668	2792	168447	4293	126876
53	226	100	33998	200	11449
54	1987	1455	84131	1701	89225
55	5035	3397	172972	4791	144996
56	3309	2981	134977	2015	137662
57	4894	4704	253931	4610	167723
58	4924	4772	242687	4648	162120
59	5255	939	186279	5085	139788
60	1988	1045	205522	1631	136995
61	4957	4291	205844	4758	140040

Tabla 56: Nodos Expandidos

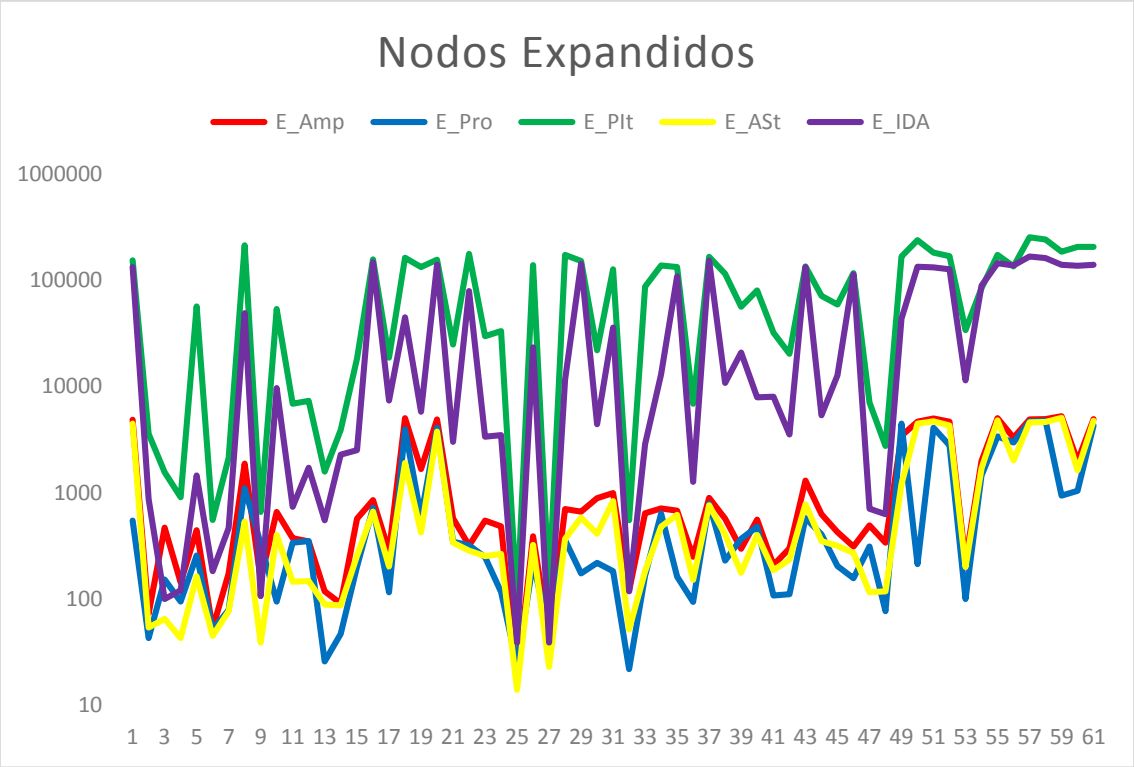

Ilustración 8: Nodos Expandidos

La tendencia en este caso es bastante similar, aunque en este caso los algoritmos de Profundidad Iterativa e IDA* presentan una diferencia mayor entre nodos generados y expandidos.

6 Presupuesto

6.1 Presupuesto Inicial

Recurso	Precio	Tiempo de vida	Tiempo uso en proyecto	Precio en el proyecto
Portátil Lenovo G505	489€	60 meses	9 meses	73,35€
Microsoft Windows 8.1	106,10€	60 meses	9 meses	15,92€
Microsoft Office Proffesional Plus 2013	539€	60 meses	9 meses	80,85€
USB Toshiba 16GB	8€	60 meses	9 meses	1,20€
Notepad++	0€	-	9 meses	0€
Python 2.7.9	0€	-	9 meses	0€
NumPy 1.9.1	0€	-	9 meses	0€
Pygame 1.9.2a0	0€	-	9 meses	0€
TOTAL				171,32€

Tabla 57: Presupuesto Inicial – Costes Materiales

Rol	Coste/Hora	Horas Totales	Coste Total
Jefe de Proyecto	49€/h	16h	784€
Analista / Programador	32€/h	280h	8690€
TOTAL			9474€

Tabla 58: Presupuesto Inicial – Costes Personales

Tipo	Coste Total
Costes Materiales	171,32€
Costes Personales	9474€
TOTAL	9645,32€

Tabla 59: Presupuesto Inicial – Total

Por tanto el presupuesto esperado inicialmente es de 9645,32€.

6.2 Presupuesto Final

Recurso	Precio	Tiempo Amortización	Tiempo uso en proyecto	Precio en el proyecto
Portátil Lenovo G505	489€	60 meses	9 meses	73,35€
Microsoft Windows 8.1	106,10€	60 meses	9 meses	15,92€
Microsoft Office Proffesional Plus 2013	539€	60 meses	9 meses	80,85€
USB Toshiba 16GB*	8€	60 meses	7,5 meses	8€
USB Lexar 32GB	12,99€	60 meses	1,5 meses	0,32€
Notepad++	0€	-	9 meses	0€
Python 2.7.9	0€	-	9 meses	0€
NumPy 1.9.1	0€	-	9 meses	0€
Pygame 1.9.2a0	0€	-	9 meses	0€
TOTAL				178,44€

Tabla 60: Presupuesto Final – Costes Materiales

*El USB Toshiba, con un tiempo de vida estimado en 5 años, dejó de funcionar a los 7,5 meses del comienzo del proyecto, por lo que no llegó a amortizarse y supuso el coste íntegro en el periodo del desarrollo. Además, tuvo que ser reemplazado por otra memoria, cubriéndose en este caso únicamente la parte proporcional a su tiempo de uso en el proyecto.

Rol	Coste/Hora	Horas Totales	Coste Total
Jefe de Proyecto	49€/h	50h	2450€
Analista / Programador	32€/h	324h	10368€
TOTAL			12818€

Tabla 61: Presupuesto Final – Costes Personales

Tipo	Coste Total
Costes Materiales	178,44€
Costes Personales	12818€
TOTAL	12996,44€

Tabla 62: Presupuesto Final – Total

El presupuesto final asciende por tanto a un total de 12996,44€.

7 Conclusiones

Una vez desarrollado todo el proyecto, ha llegado el momento de analizarlo en su conjunto, comprobando si se han cumplido los objetivos fijados en el apartado *Objetivos* y analizando los resultados obtenidos en la sección *Pruebas y Resultados*.

En primer lugar, todos los objetivos propios del proyecto han sido alcanzados. Se ha logrado implementar un juego de Sokoban, los distintos algoritmos con técnicas de Inteligencia Artificial que generan soluciones que constan de una serie de pasos para llegar desde el estado inicial hasta el estado final y también se ha cumplido con el objetivo de desarrollar una Interfaz Gráfica que permite jugar con los niveles de Sokoban y visualizar las soluciones halladas por los algoritmos.

Respecto al objetivo de medir la calidad de los algoritmos en función de distintos parámetros también se puede decir que se ha cumplido. En el apartado de *Pruebas y Resultados* se han comparado los distintos algoritmos según las soluciones generadas. Así, ha podido observarse que aunque el Algoritmo de Primero en Profundidad es el que más niveles resuelve, no asegura optimalidad, por lo que conviene considerar otros algoritmos. De este modo se ve que el siguiente algoritmo que más problemas resuelve es A*, que es óptimo por emplear una heurística admisible y que además de entre los algoritmos que aseguran optimalidad es el que para la mayoría de problemas emplea un menor tiempo. De este modo, una solución combinada seria ejecutar A* para encontrar soluciones óptimas, y en caso de no encontrarse, emplear Búsqueda en Profundidad para tratar de obtener al menos una solución no óptima.

También se realizaron comparativas de nodos generados y expandidos, aunque su importancia es algo menor que la mencionada en el párrafo anterior. No se realizaron finalmente estudios de uso de memoria, pero por la propia definición de los algoritmos se sabe que los que mayor uso de memoria realizan son Amplitud y A*, siendo en este caso aún mayor para el algoritmo de Amplitud por tener una lista de nodos generados para evitar repetidos. De este modo, si hubiera que elegir un algoritmo por su menor uso de memoria y además que dé soluciones óptimas, la opción a elegir sería IDA*, ya que resuelve más tableros que Profundidad Iterativa.

Respecto a los objetivos personales que quería alcanzar con el proyecto, también se pueden considerar alcanzados. Tras la realización del proyecto puedo considerar que manejo el lenguaje Python con cierta soltura. Además también he podido realizar una interfaz gráfica para el programa, que era otro de los objetivos, y aunque el dominio de esta parte no es tan elevado como el logrado con Python, se puede considerar que se han adquirido unos conocimientos que permitirán seguir mejorando.

8 Líneas Futuras

Son muchas las expansiones y mejoras que se pueden realizar en este proyecto o en otros que se centren en un tema similar, ya que este es un campo de estudio muy amplio. Muchas son las ideas que has surgido después de haber finalizado el proyecto. Hay una innumerable cantidad de algoritmos que podrían dar buenos resultados en un problema como este. O podrían ejecutarse técnicas de Aprendizaje Automático, como el Aprendizaje por Refuerzo con técnicas de Clustering o el Aprendizaje Incremental, de modo que se desarrollaran agentes que aprendieran a partir de una gran colección de instantes de juego y así decidir cuál es el mejor movimiento a realizar en una situación dada.

Otra posible funcionalidad extra que podría diseñarse para proyectos futuros sería la de permitir la creación de distintos tableros de juego, ya se manualmente o automáticamente a partir de algunos parámetros. Sería además interesante estudiar distintos modos para integrar la parte de ejecución de los algoritmos en la interfaz gráfica, para que de este modo esté el proyecto más unificado y todas las funcionalidades se puedan acceder desde el mismo programa. Tal vez esta parte podría abrirse en una nueva ventana sin bucle de juego para poder seguir trabajando con la interfaz gráfica o se podrían estudiar otras formas de desarrollar la parte gráfica, como Panda3D.

También se ha tomado muy en cuenta la posibilidad de crear una versión del sistema en Python 3, ya que está bastante extendido.

Tras realizar distintas pruebas con las dos partes del programa, se percibió que había algunas características que se podían mejorar para hacer más cómodo su uso a los usuarios. Por ejemplo, en el menú de selección de niveles se debería incluir alguna opción para volver al menú principal o en las partidas se podría incluir alguna funcionalidad para volver al menú o reiniciar la partida en caso de caer en una situación de Deadlock que no permita acabar el juego, ya que en la versión actual la única solución es cerrar la interfaz. También podría estudiarse el control de menús por medio del ratón.

Respecto a la parte del programa de ejecución de algoritmos, también hay muchas mejoras posibles. Un ejemplo sería la de permitir al usuario decidir si quiere sólo resolver algún nivel concreto y con unos algoritmos deseados o poder fijar de forma simple el tiempo máximo de ejecución de los algoritmos de búsqueda, que actualmente sólo es posible modificando el código.

Quizás una mejora interesante sería la de generar ejecutables de Windows para que usuarios que no cuenten con Python 2.7 y los paquetes empleados en el proyecto puedan ejecutar los archivos sin necesidad de realizar ninguna instalación. En esta ocasión no se realizó debido a que muchos programas generadores de ejecutables no soportan las librerías empleadas y/o no permiten proyectos con varios archivos de Python, aunque con un estudio en profundidad podría ser posible adaptar estos programas o generar los ejecutables a mano.

9 Bibliografía

[1] J. C. Culberson, *Sokoban is PSPACE-Complete,* Edmonton, Alberta (Canada): University of Alberta - Department of Computing Science, 1997.

[2] R. Kaye, «Minesweeper is NP-complete,» *Mathematical Intelligencer,* vol. 22, nº 2, pp. 9-15, 2000.

[3] D. Ratner y M. K. Warmuth, «Finding a shortest solution for the N*N-extension of the 15-puzzle is intractable,» *J. Symbolic Comp,* nº 10, pp. 111-137, 1990.

[4] «Python 2.7.10 Documentation,» Python Software Foundation US, [En línea]. Available: https://docs.python.org/2/index.html. [Último acceso: 13 Septiembre 2015].

[5] A. Jughanns, *Pushing the Limits: New Developments in Single-Agent Search,* Edmonton, Alberta (Canada): University of Alberta - Department of Computing Science, 1999.

[6] «Pygame,» [En línea]. Available: http://pygame.org/wiki/about. [Último acceso: 22 Septiembre 2015].

[7] «Sokoban Perfect Plus,» Thinking Rabbit, [En línea]. Available: http://sokoban.jp/. [Último acceso: 20 Septiembre 2015].

[8] E. Sever, «Sokoban·Erim,» [En línea]. Available: http://www.erimsever.com/sokoban8.htm. [Último acceso: 26 Septiembre 2015].

[9] Y. Chang, «Sokoban.org,» 13 Julio 2013. [En línea]. Available: http://sokoban.org/about_sokoban.php. [Último acceso: 20 Septiembre 2015].

[10] Departamento de Ingeniería Informática - UC3M, *Apuntes de Teoría Avanzada de la Computación,* Leganés, Madrid (España).

[11] M. Fryers y M. Greene, «Sokoban,» *Eureka,* nº 54, 1995.

[12] S. Russel y P. Norwig, Artificial Intelligence: A Modern Approach (Third Edition), Upper Saddle River, Nueva Jersey (EEUU): Prentice Hall, 2010.

[13] T. Harris, «yUML,» [En línea]. Available: http://yuml.me/. [Último acceso: 25 Septiembre 2015].

[14] C. W. Dawson y G. Martín Quetglás, El Proyecto Fin de Carrera en Ingeniería Informática: Una Guía para el Estudiante, Prentice Hall, 2002.

10 ANEXOS

10.1 ANEXO I: Summary

10.1.1 Introduction

Sokoban is one of the most interesting board-games because of the diversity of possibilities that it brings to the investigation. That's the reason why this project will try to show a little part of all that this game embraces.

The intention at this introductory paragraph is to present the motivation for doing this project, its different stages and a keyword section with common words used in this document. It will finish detailing the used organization for this report.

10.1.1.1 Motivation

Over the last decades, Artificial Intelligence has obtained a big quantity of uses to solve several problems, both in investigation and in daily life. It's possible to find videogames that use Artificial Intelligence to model characters' behaviors or to find the best movement path. We can find Artificial Intelligence in predictive search or in adverts' suggestions when surfing the net, as well as in data analytics. And in today's society, it doesn't seem strange to hear a machine answering the phone when calling to technical support.

Artificial Intelligence or AI, it's a branch of Computer Science that it's usually defined as the science that it's on charge to give intelligence to machines or as the study and design of Intelligent Systems. This discipline has turned into a very important element nowadays, especially in Industrial, Technologic and Business fields. And all this is because AI can solve some of the most difficult existing informatics problems.

One of this difficult problems is Sokoban. This game has been demonstrated that it’s in the complexity class PSPACE-Complete [1], that it’s believed to be a more complex class than the NP-Complete group of problems, where there are some problems about other puzzle games, like Minesweeper's Consistency [2] and the search of the minimum number of movements in 15 puzzle [3].

Sokoban is a game where typically there is only one agent or player and where de character can push some blocks over the game board, but he's only allowed to push a crate at a time. There are some special squares, called goals, where the blocks have to be placed to win the game.

This investigation's intention is to implement different search algorithms capable of finding solutions to this popular game and to analyze each of the used techniques to make a comparison between them.

In addition, it's intended the design a graphic interface so that way users can also play with the different levels and to be able to visualize the solutions found by the search algorithms.

This game is also a personal motivation, because Sokoban is a game that I discovered while I was in my first year of the degree during the Programming subject, and it helped me to learn the basics contents of programming languages generics and Java language particulars.

The final assignment of the subject consisted on two phases. The first one consisted on implementing the game as a printing of characters in the screen, using some basic functions and structures. At second phase we used more advanced methods and classes and we used a graphic interface created by our teacher.

That was the moment I fell in love with everything about the game and I felt a little sorry for not learning at that time how to create the graphic design with a programming language.

This End-of-Degree Project means to me the perfect opportunity to go back to the game that marked the start of my degree, so this is the perfect chance to include a graphic interface that at that time I considered out of my range and, at the same time, this is a opportunity to learn a new programming language. If at that time my Java knowledge was reduced to some little assignments over the subject duration, this time the project will help me to learn Python from scratch.

10.1.1.2 Project Stages

Before starting the development of the graphic interface and the solvers stated in the previous section, it's mandatory to present a basic view of the game, namely, it's necessary to define the different movements allowed and the rules that compound it. It has to be checked that all the components are working properly, and then a more colorful interface can be created and the different solvers can be implemented.

The Project stages are the following ones:

- **Stage 1**: Aim definition. The desired goals of the project's realization were stated.
- **Stage 2**: Investigating about Python 2 [4], learning its basic operation before starting to work with the language.
- **Stage 3**: Basic definition of the Sokoban game. The rules that control a simple Sokoban level written in a text file were defined in Python. The instance represents every square with an ASCII character, and the instance is showed in a command line, asking for a letter input (W, D, X, A) that represents the different movements realized. The program checks if the movement is valid, in which case the instance is modified by the games rules and displayed in the screen, or in case that the movement can't be accepted, it shows a message stating the reason (an invalid key or a movement not allowed).
- **Stage 4**: search of the levels that will be used in this research. Some Sokoban levels that could be used to solve were searched. It was decided to use the ones that Andreas Jughanns used in his PhD thesis [5], the Kids Problems' set. This and other sets over the internet are usually written in a special ASCII notation called XSB Format, so the rules had to be adapted so they could work with the new format. The Original file had all the Kids Problems in the same archive, so the file had to be cleaned and divided into 61 independent files, one for each level, with a Python program created for this purpose.
- **Stage 5**: after the search of level was over and after learning about the format notations at the previous stage, it was decided to continue investigating in that field and learn more about the specifics of the Sokoban game domain, and so the State of the Art section was written. The Initial Budget was written too.
- **Stage 6**: game in graphic mode. After studying about Pygame programming and practicing a little with Pygame [6], the implementation of the first graphic representation was realized, game loops were defined, and it was designed the key reading events and the different functionalities that use the previously designed rules.

- **Stage 7**: Solver design. At this stage the search algorithm that will solve the puzzles were implemented and some improvements were added, reducing the execution times and the expanded and generated nodes.
- **Stage 8**: A complete menu design. As the final part of the implementations, a complete menu was created for the graphic interface, allowing to play the 61 levels and showing the results of the search algorithms.
- **Stage 9**: drafting of the remaining parts of the document.

10.1.1.3 Keywords

- **AI:** it's the acronym of Artificial Intelligence. It refers to the branch of Computer Science that studies the development of systems that solve problem autonomously.
- **Deathlock:** situation at a Sokoban game where it's impossible to win the play by performing any of the allowed actions in the rules.
- **Kids Problems:** common name of the "Dimitri and Yorick" set of Sokoban instances designed by Jacques Duthen. It's composed for 61 Sokoban game boards and it was created with children from 4 to 8 years old in mind.
- **LURD Notation:** it's the most popular representation format for movements in Sokoban solutions.
- **Numpy:** it's a Python library that works with data, numbers and arrays.
- **Pygame:** Python library that helps in the graphics design of games and graphic interfaces written in this programming language.
- **Python:** it's an interpreted programming language whose philosophy emphasizes a very readable code and where indentation has a lot more importance than in other programming languages.
- **XSB Notation:** representation format for Sokoban levels where a specific combination of ASCII characters is used.

10.1.1.4 Document Organization

The Project structure will be the following one:

1. **Introducción (Introduction):** at the introduction section, that we are currently reading, it's stated a first view of the document, talking about the motivation behind this work realization. The project stages will be defined too, as well as some keywords and the organization used in this document.
2. **Estado del Arte (State of the Art):** in the second chapter of this research all the situation that has relation with the problem will be introduced, talking about the game as well as different solver techniques used in Artificial Intelligence.
3. **Objetivos (Goals):** this paragraph will present, as its name states, the objectives that are expected to achieve while working in the project.
4. **Desarrollo (Development):** this section, that it's assumed to be the core of the investigation, will verse about all the steps realized in this project, defining the different versions of the algorithms, the modifications made and the decision making. It will display all the part in charge of the level solvers, as well as the game graphic interface.
5. **Pruebas y Resultados (Testing and Results):** in the Testing chapter all the used algorithms will be checked and the obtained results will be analyzed.
6. **Presupuesto (Budget):** this section will keep a summary of the different costs related of the developing of the project.
7. **Conclusiones (Conclusions):** this paragraph will collect all the data obtained after the project realization and it will be tested if the objectives has been achieved.
8. **Líneas Futuras (Future Improvements):** it will be stated some additions or modifications that could be made in future works or versions of the project.

To end this document, a pair of sections for *Anexos* (Appendices) and *Bibliografía* (Bibliography) will be added, where additional information about the mentioned contents and the information sources will be included.

10.1.2 Corpus

10.1.2.1 Chosen Decisions

When implementing the different algorithms, the first decision made was the programming language used. It was decided to use Python, being a language that is not used in any mandatory practice of the degree and I had no previous knowledge of the language. This language caught my attention when in a subject were proposed several optional practices, one of which used some simple Python. By then I was already planning the language to be used in the project, with the main candidates beeing C ++ (of which I have basic knowledge) and Python (which I knew nothing). I took that opportunity to learn the language and I really liked Python, so I decided to use it.

After opting for Python, I had another decision to make, and was to use Python 2 or Python 3. After some research I chose Python 2, since it is the most documented and extended version, but Python 3 is gaining strength and is also quite widespread.

Moreover, it was decided to use the library Numpy to facilitate reading data file with a specific format and to easily work with them a two-dimensional array representing the game board.

The next decision was to choose the method to develop the graphical interface of the game, going for Pygame. This set of modules for Python is based on the SDL library and allows you to create games and multimedia programs [6].

When executing the various algorithms, as some may not find solutions or spent too long times, a maximum execution time was of 30 minutes was decided.

Regarding the search algorithms used for finding solutions in different game boards, they ones being used are in breadth first search, depth first search, iterative deepening depth first search, A * and IDA*. These algorithms have different kinds of improvements to reduce search times so they become better in finding solutions.

Breadth first search and Depth first search, to prevent the generation of repeated statements that can create loops (especially in the case of depth search, these loops can prevent the algorithm to end) a list of generated states has been created. This way, only the new nodes are considered and avoids falling into cycles.

On the other hand, a function which finds a large number of Deathlocks has been defined, thereby avoiding to expand nodes that will never reach a solution. Specifically, this function checks stones with walls on two adjacent sides (at a corner) or the case in which two or more adjacent boxes are next to a wall.

These modifications managed to improve success in finding solutions rate and the time spent to find solutions dropped a high magnitude. Before the improvement of the list of generated nodes, there were some cases (Test 24) that didn't find solutions in breadth and depth search with a 10-minute limit and with this improvement the algorithms found solutions in 64 and 15 seconds respectively. In addition, when used the Deadlocks detection, a larger number of solutions was found with all methods (Iterative depth in Test 24 could find no solution in 10 minutes, and with the Deadlocks detection finds a solution in a little more than 5 minutes) and reduced the time to find solutions, becoming about 6 times lower (Test 24: A* lowers from 39 to 7 seconds and IDA * goes from 182 to 30. Breadth and depth get 18 and 2 seconds, respectively).

For Informed search algorithms have been considered a heuristic that calculates the distance of each stone to the nearest base, adds this distances and then adds the distance from the player to the farthest box. It is also considered whether the board is in a Deadlock position, adding a value of 1 million if there is a deadlock or 0 otherwise. This heuristic is admissible because it never overestimates the cost to reach the goal from a particular state:

- For distances to the bases, at the best case all the blocks end in the base that used to calculate the distance, being equal to the expected cost. In other cases, they end up on some other goal, so it will have a cost equal to or greater than calculated, so that the heuristic will always have a lower value.

- Considering the distance from the player to the farthest stone to solve the board, the player has to take all the stones to the bases, so sooner or later the player must go to the farthest stone to move it, so this part is admissible too.

- Finally, the added condition for Deadlock is admissible too, because if there is a Deadlock, it doesn't add any additional cost to the calculated, so it is still admissible, and if there is a Deadlock, the solution will never be reached (can be considered infinite cost) so adding 1000000 is still lower than the actual cost.

It was intended that part of the search for solutions was called from within the GUI, but the operation spent too much time without calling the game loop algorithms while seeking solutions and gave some problems.

10.1.3 Conclusions

Once developed the entire project, it is time to analyze it as a whole, checking if the objectives set out in paragraph *Objetivos (Goals)* has been achieved and analyzing the results of the *Pruebas y Resultados* (Tests and Results) section.

First, all the explicit project objectives have been achieved. It has been implemented a game Sokoban, different algorithms that use Artificial Intelligence techniques to generate solutions that consist of a series of steps to get from the initial state to the final state and has also met the goal of developing a GUI that allows to play Sokoban levels and visualize the solutions found by the algorithms.

Regarding the objective of measuring the quality of the algorithms based on various parameters, it can be said that this part has also been completed. In the area of *Pruebas y Resultados* (Testing and Results) various algorithms were compared according to generated solutions. Thus, it has been observed that although the depth-first algorithm solves more levels, it doesn't ensure optimality, so it'is appropriate to consider other algorithms. The second algorithm that solves most problems is A *, which assures optimal solutions because it uses an admissible heuristic and also, in the list of algorithms that ensure optimality, it's the one that employs a shorter time. Thus, a the best algorithm could be to use A* to find optimal solutions, and if not found, use depth first search to try to get at least a non-optimal solution.

Comparisons were also made of nodes generated and expanded, though their importance is somewhat lower than that mentioned in the preceding paragraph. Finally, memory usage studies were not performed, but the definition of algorithms that are known to carry higher memory usage are Breadth First Search and A*, and in this project the memory used for Breadth First Search is even greater because it uses a list of generated nodes to avoid repeated states. Thus, if I had to choose an algorithm for its lower memory usage and that assures optimal solutions I would choose IDA *, because it solves more problems that the other option, Iterative Deepening Depth First Search.

Regarding the personal goals I wanted to achieve with the project, they can also be considered achieved. After the project I can say that I can use the Python language with some ease. In addition, I have also been able to make a GUI for the program, which was another objective, and although mastering this part is not as high as that achieved with Python, I can consider to have acquired knowledge that will allow further improvement.

10.2 ANEXO II: 61 Kids Problems

En este anexo se recogen los 61 problemas del conjunto Dimitri-Yorick tal y como son generados por la interfaz gráfica del programa desarrollado.

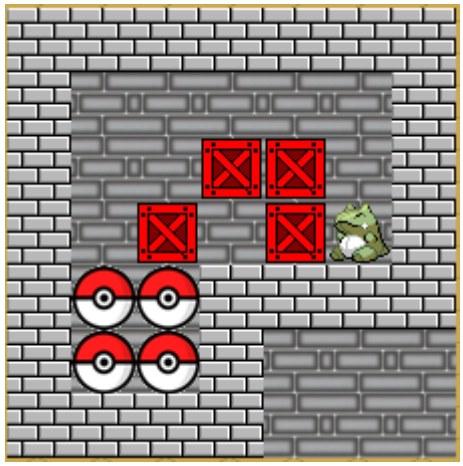

Ilustración 9: Nivel 1

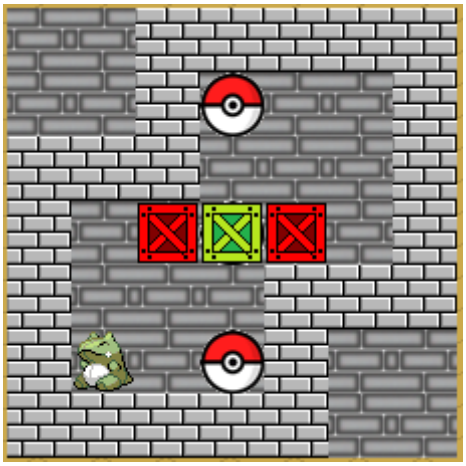

Ilustración 10: Nivel 2

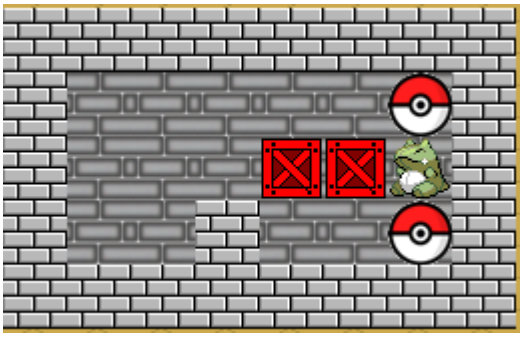

Ilustración 11: Nivel 3

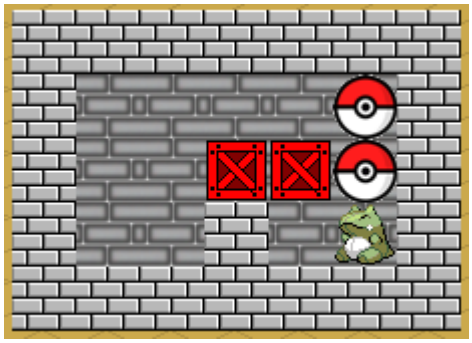

Ilustración 12: Nivel 4

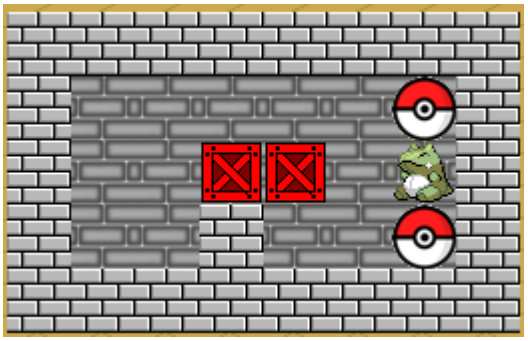

Ilustración 13: Nivel 5

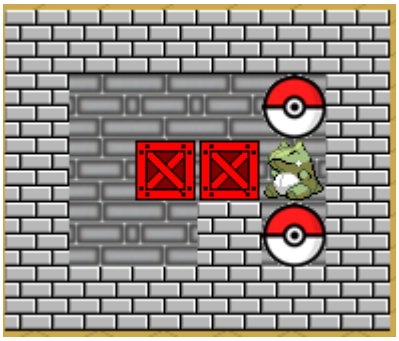

Ilustración 14: Nivel 6

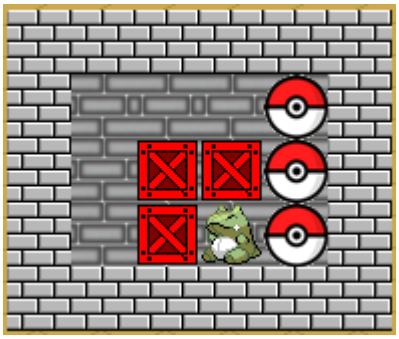

Ilustración 15: Nivel 7

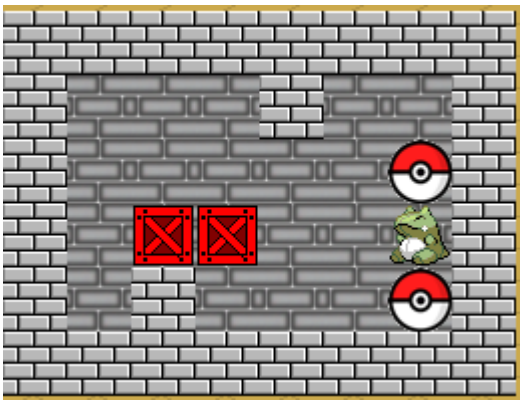

Ilustración 16: Nivel 8

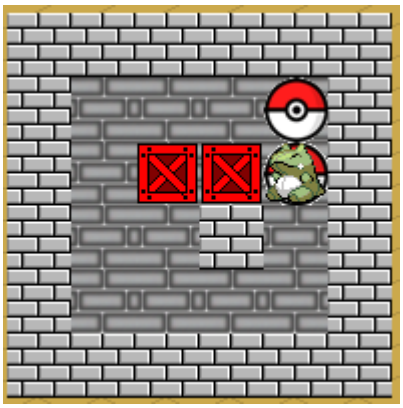

Ilustración 17: Nivel 9

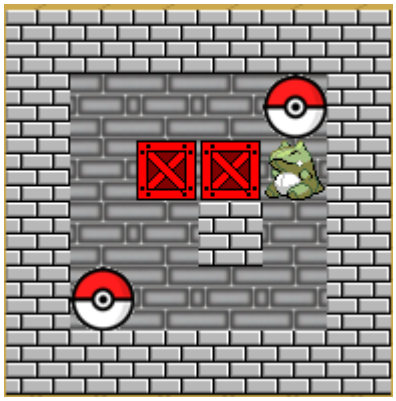

Ilustración 18: Nivel 10

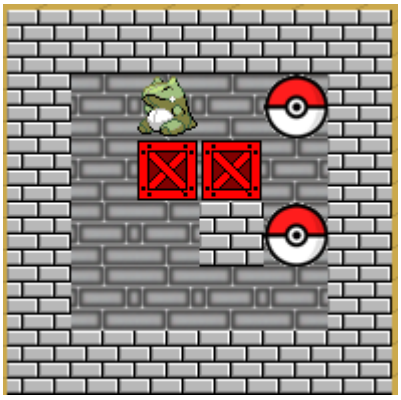

Ilustración 19: Nivel 11

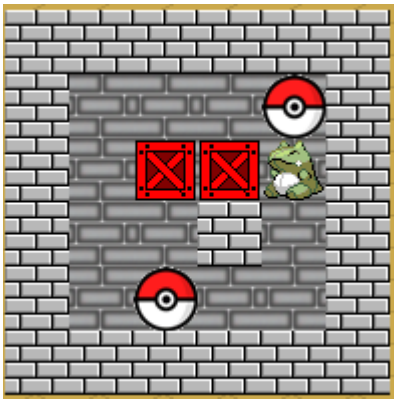

Ilustración 20: Nivel 12

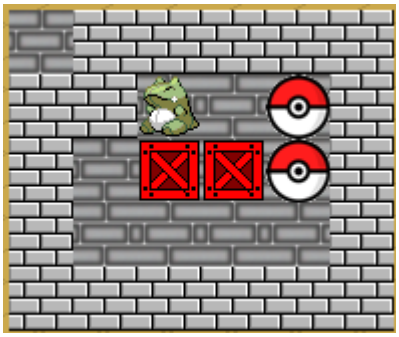

Ilustración 21: Nivel 13

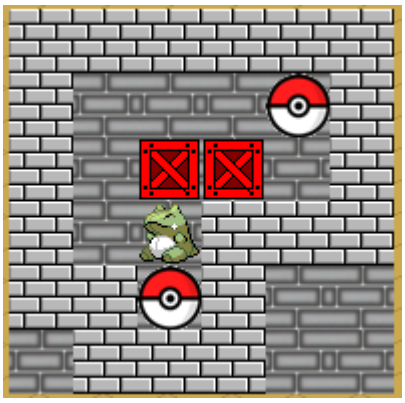

Ilustración 22: Nivel 14

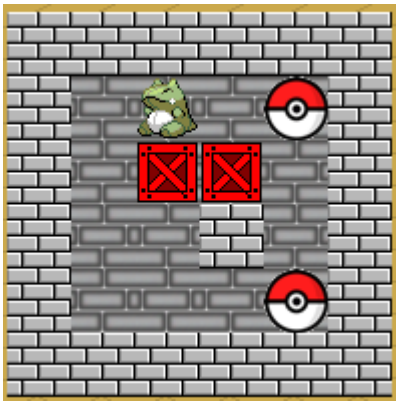

Ilustración 23: Nivel 15

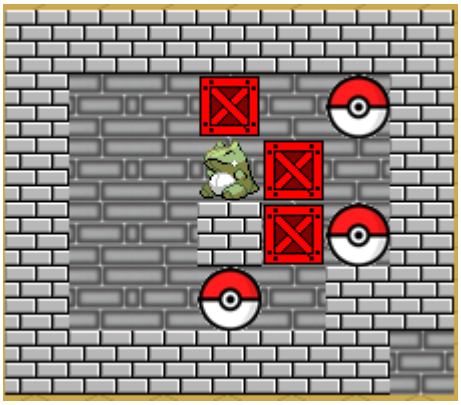

Ilustración 24: Nivel 16

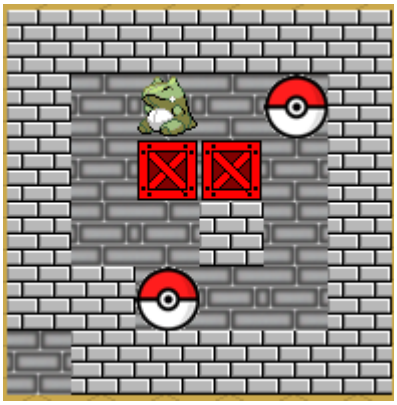

Ilustración 25: Nivel 17

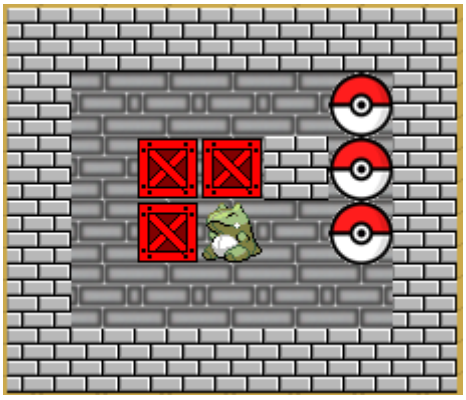

Ilustración 26: Nivel 18

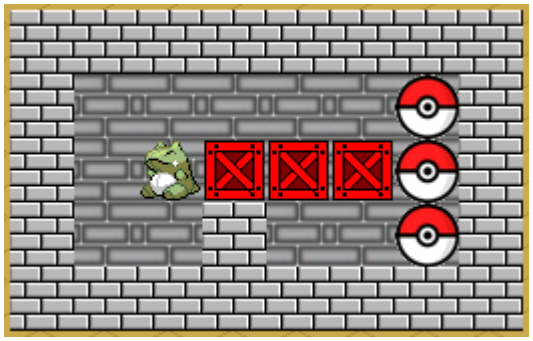

Ilustración 27: Nivel 19

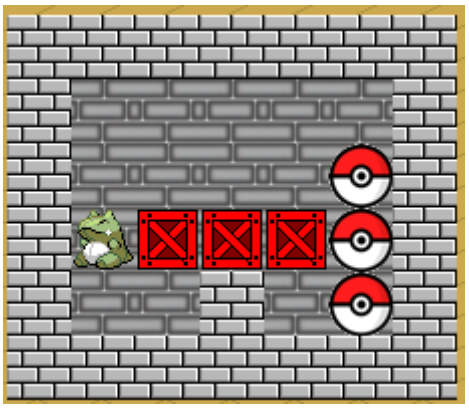

Ilustración 28: Nivel 20

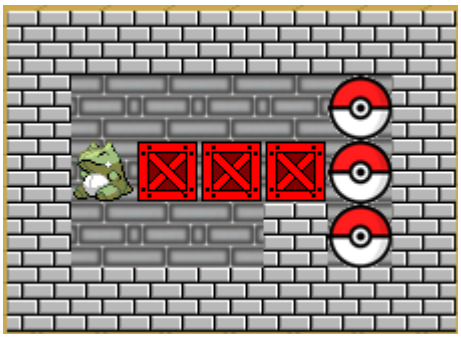

Ilustración 29: Nivel 21

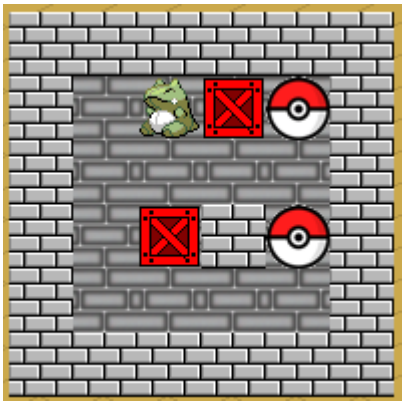

Ilustración 30: Nivel 22

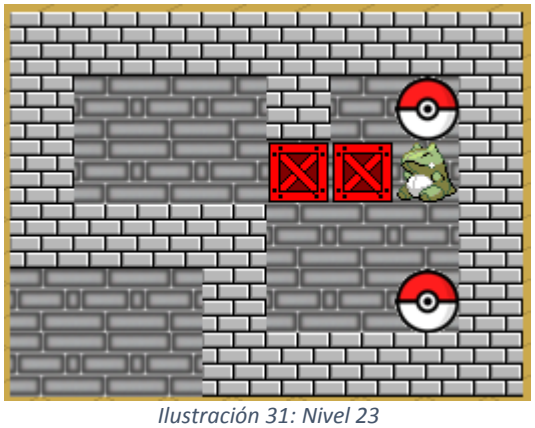

Ilustración 31: Nivel 23

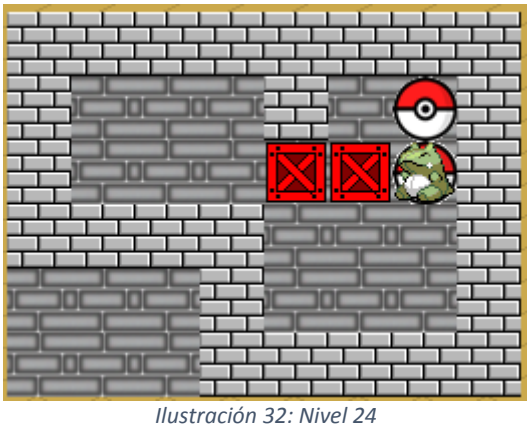

Ilustración 32: Nivel 24

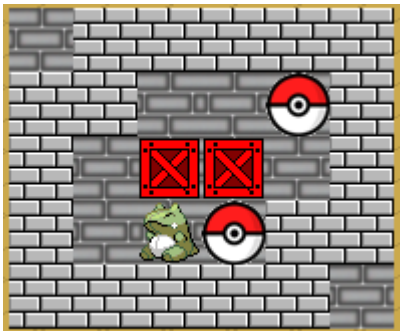

Ilustración 33: Nivel 25

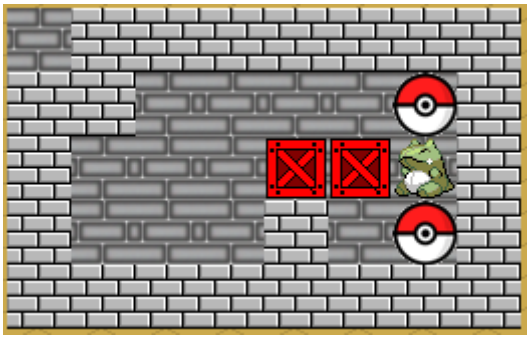

Ilustración 34: Nivel 26

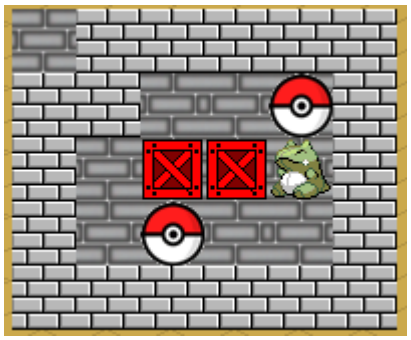

Ilustración 35: Nivel 27

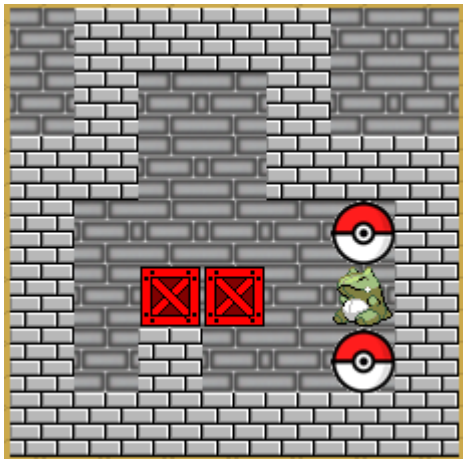

Ilustración 36: Nivel 28

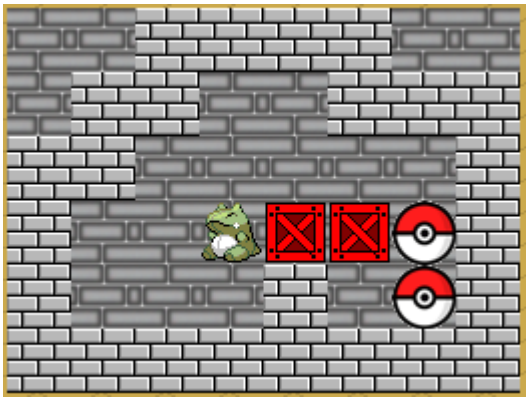

Ilustración 37: Nivel 29

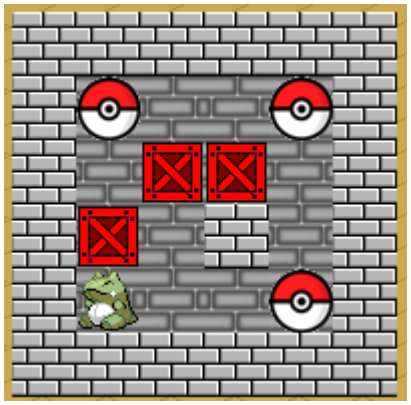

Ilustración 38: Nivel 30

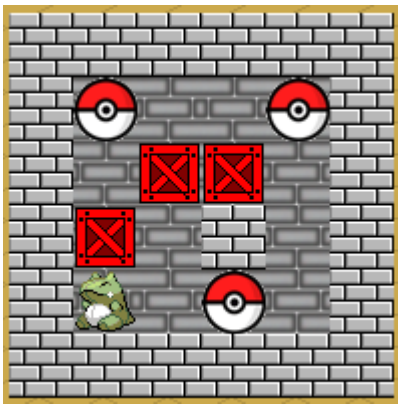

Ilustración 39: Nivel 31

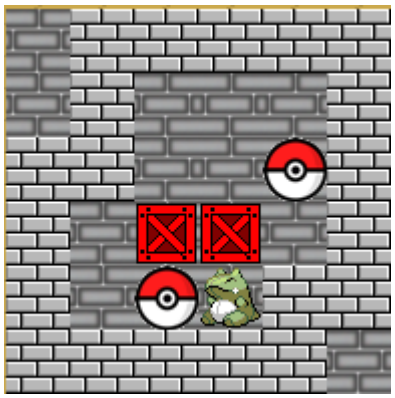

Ilustración 40: Nivel 32

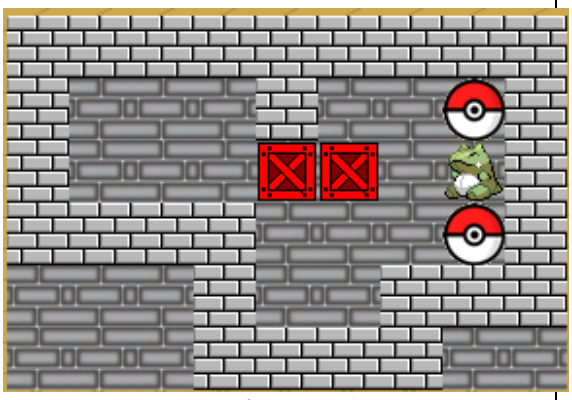

Ilustración 41: Nivel 33

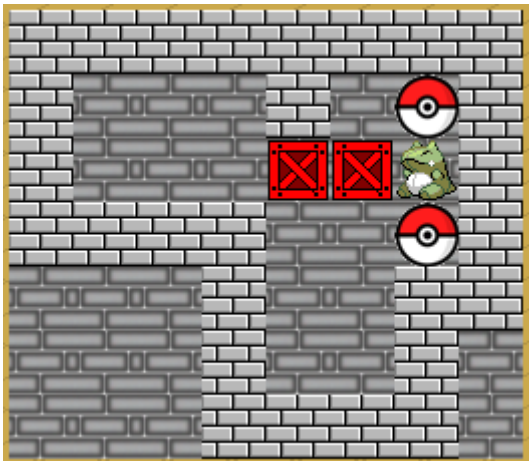

Ilustración 42: Nivel 34

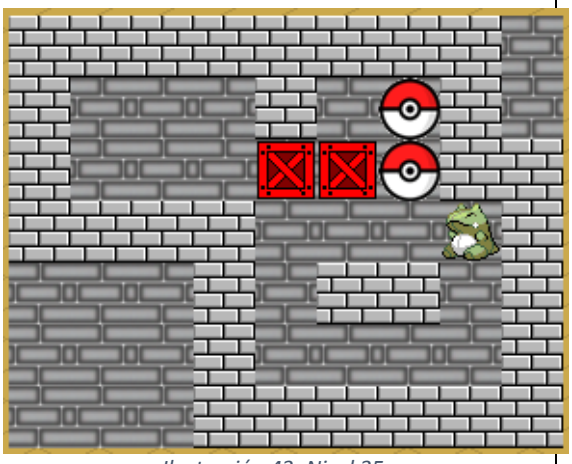

Ilustración 43: Nivel 35

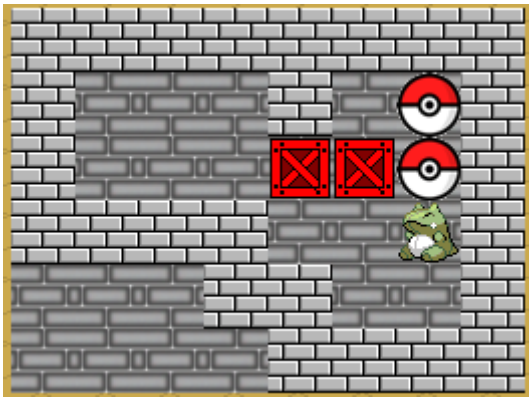

Ilustración 44: Nivel 36

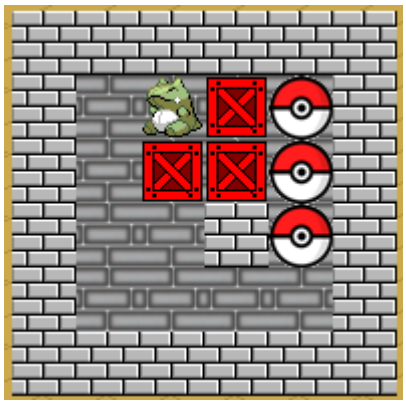

Ilustración 45: Nivel 37

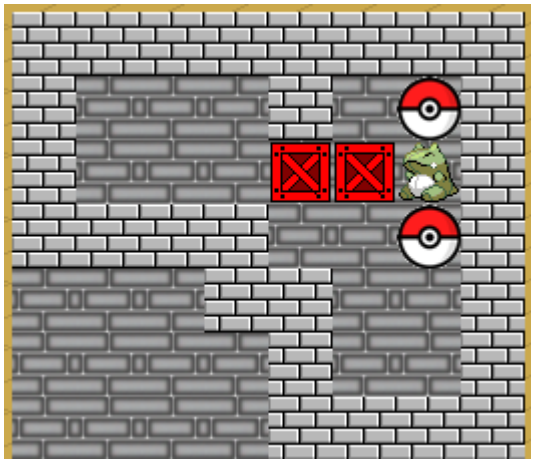

Ilustración 46: Nivel 38

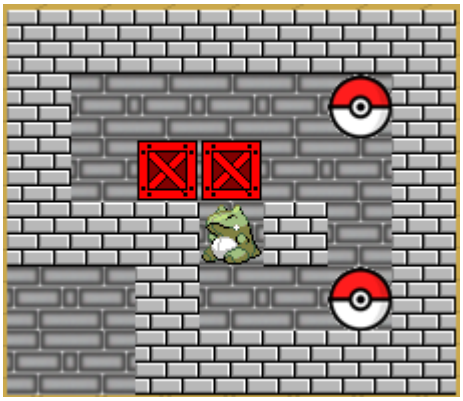

Ilustración 47: Nivel 39

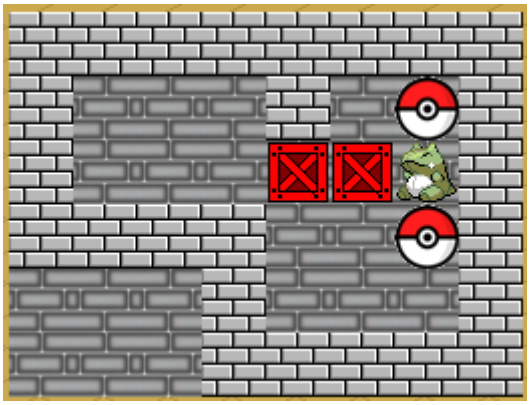

Ilustración 48: Nivel 40

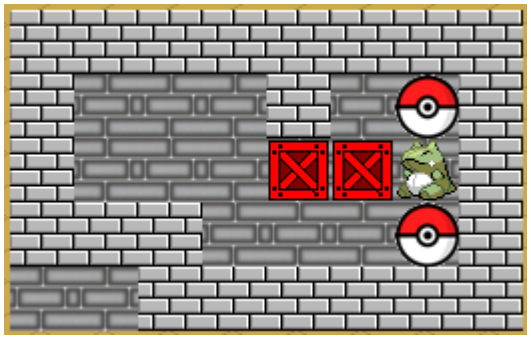

Ilustración 49: Nivel 41

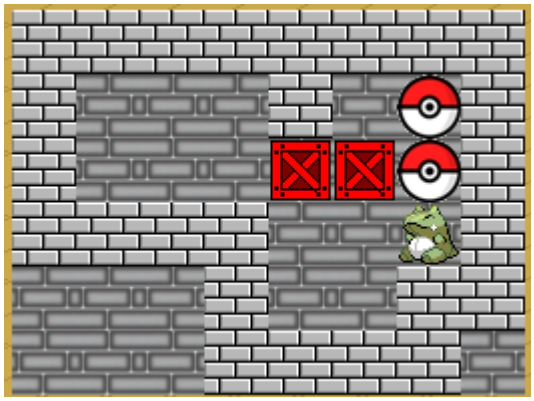

Ilustración 50: Nivel 42

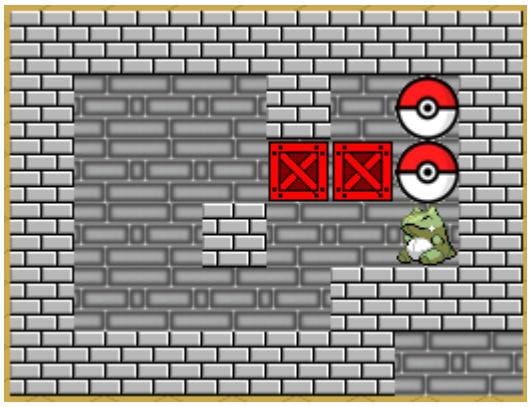

Ilustración 51: Nivel 43

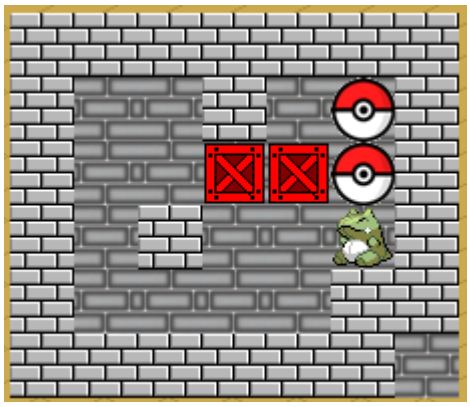

Ilustración 52: Nivel 44

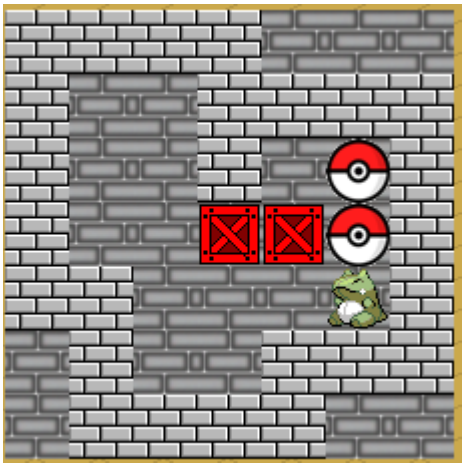

Ilustración 53: Nivel 45

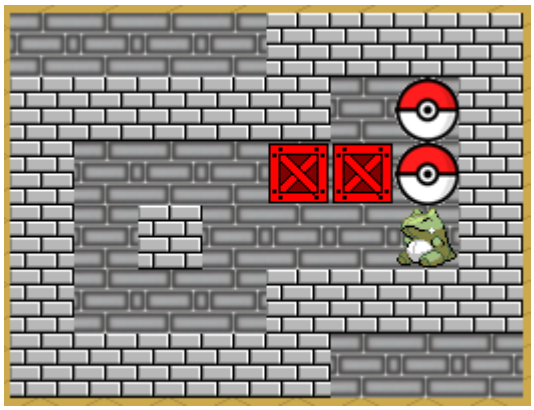

Ilustración 54: Nivel 46

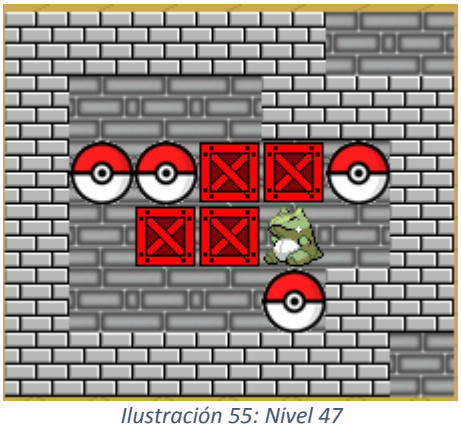

Ilustración 55: Nivel 47

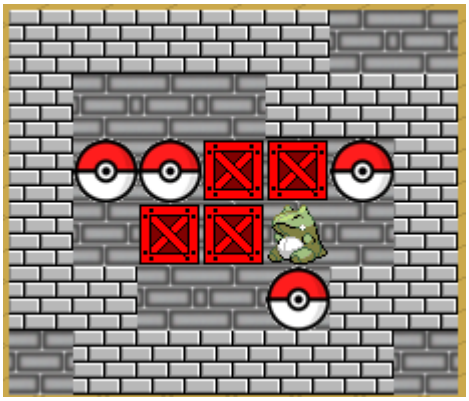

Ilustración 56: Nivel 48

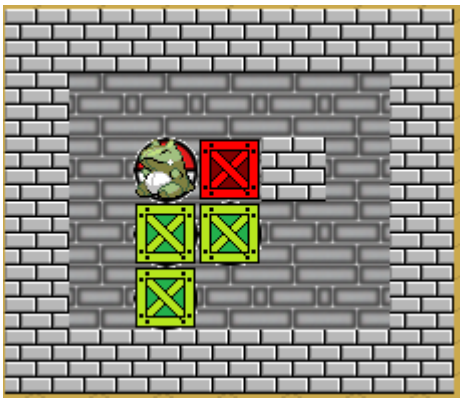

Ilustración 57: Nivel 49

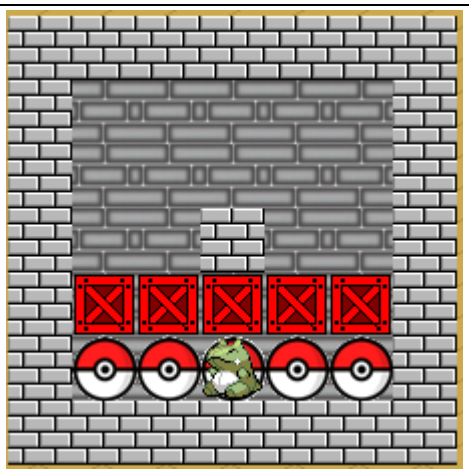

Ilustración 58: Nivel 50

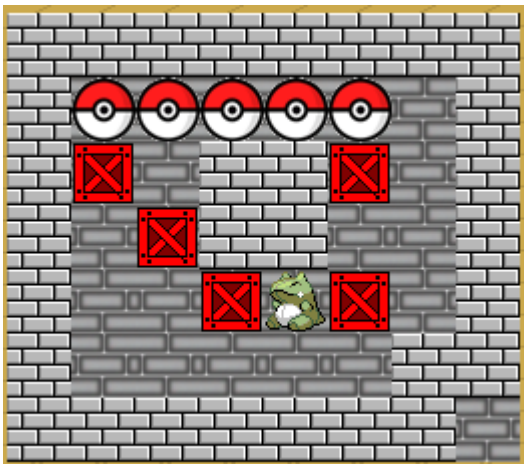

Ilustración 59: Nivel 51

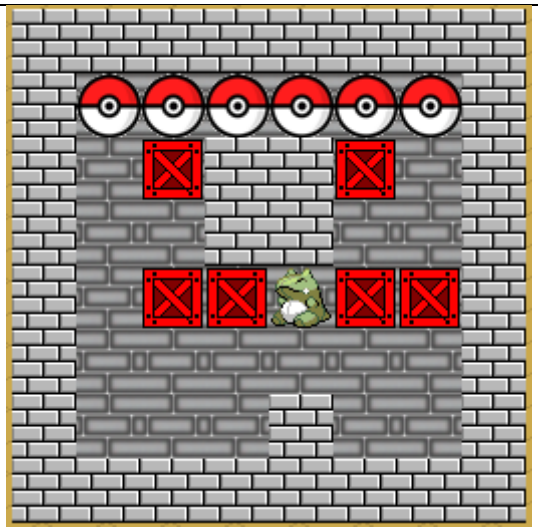

Ilustración 60: Nivel 52

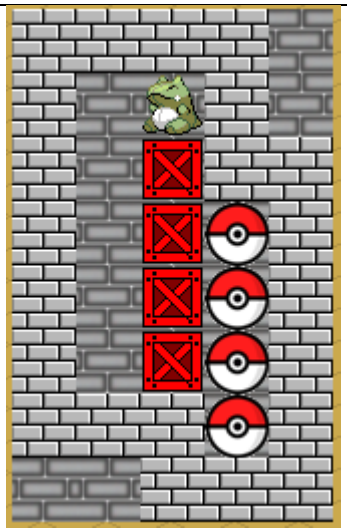

Ilustración 61: Nivel 53

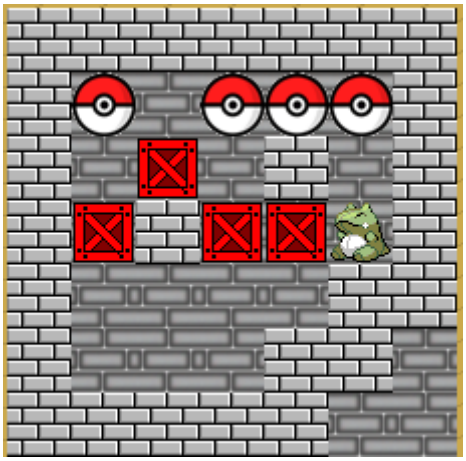

Ilustración 62: Nivel 54

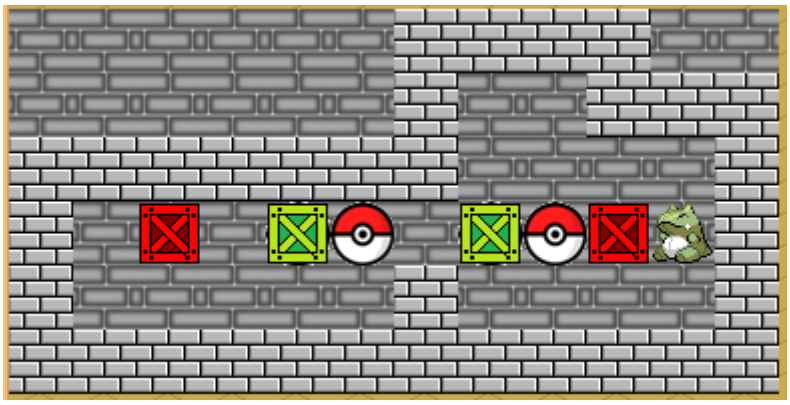

Ilustración 63: Nivel 55

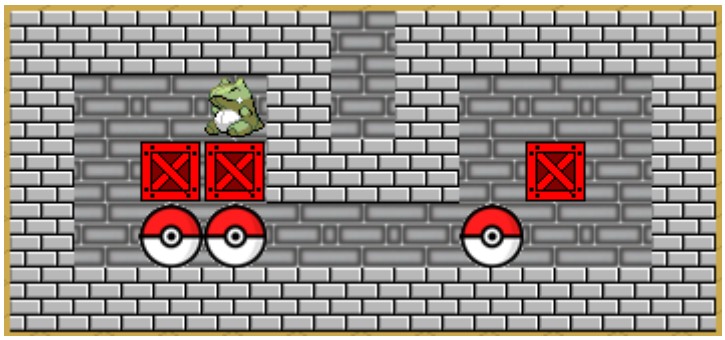

Ilustración 64: Nivel 56

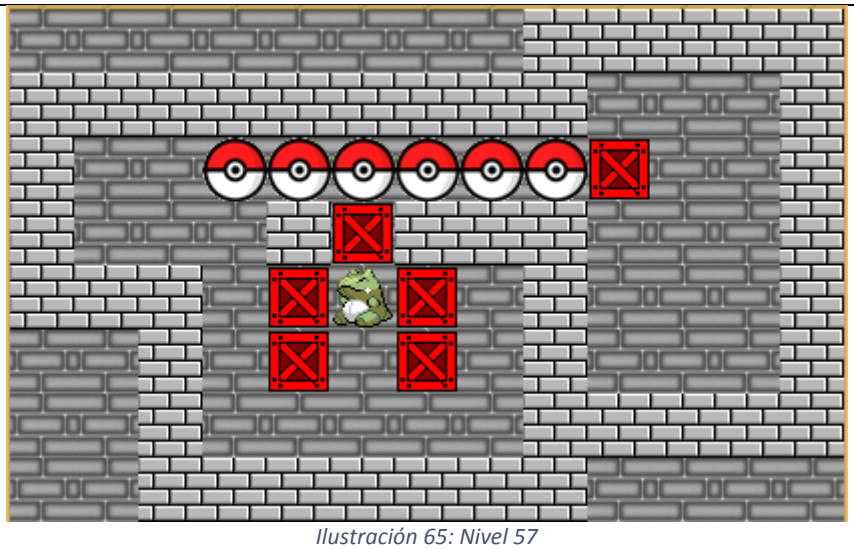

Ilustración 65: Nivel 57

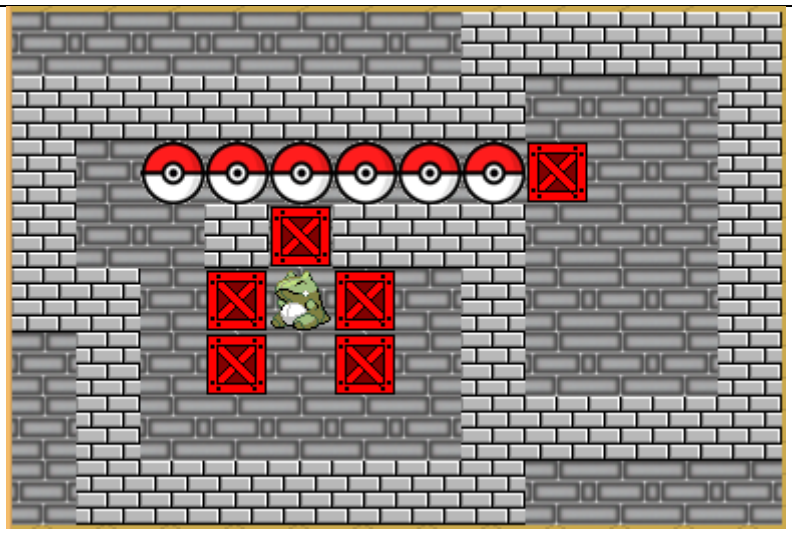

Ilustración 66: Nivel 58

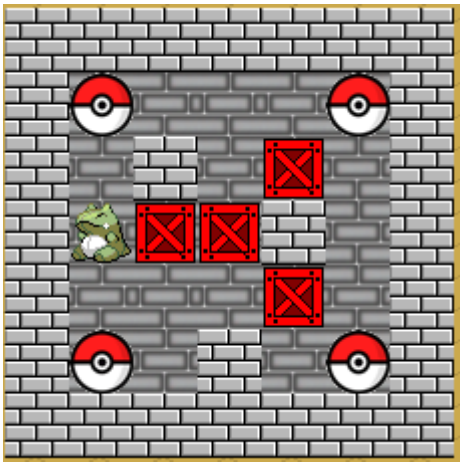

Ilustración 67: Nivel 59

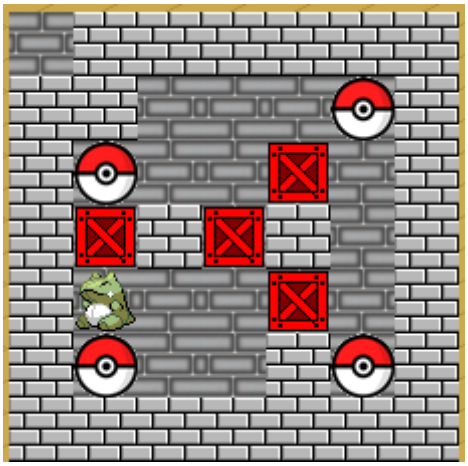

Ilustración 68: Nivel 60

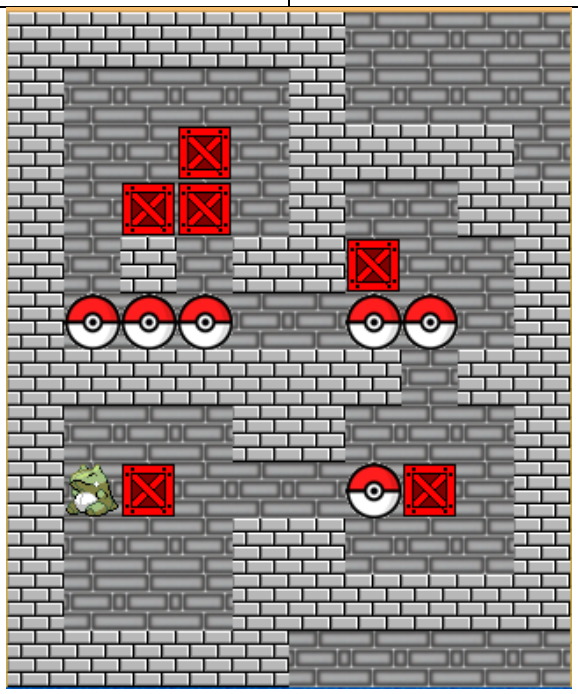

Ilustración 69: Nivel 61

Tabla 63: 61 Kids Problems

10.3 ANEXO III: Resultados Completos

10.3.1 Amplitud

Nivel	S_Amp	P_Amp	C_Amp	T_Amp	G_Amp	E_Amp
1	None	0	0	1800.059832	5529	4873
2	urrUUrrdLulDDllu R	17	6	0.863171559	75	73
3	dllUluRRDldR	12	5	18.79831147	564	470
4	lULuRllldRRR	12	6	2.138465127	163	144
5	dllULuRllldRRRuR DldR	20	9	17.04847476	483	445
6	ullldrURurDlluR R	18	5	0.557234418	58	53
7	ruullddRRlURlluR R	18	6	3.418728939	197	175
8	dlllUruLulddRRRll luRRRRDldR	28	11	285.7781213	2009	1890
9	ddlluURlluRR	12	4	2.568935388	190	153
10	ullDDuurrdLrddlL uUluRR	22	7	34.643381	715	661
11	lddrURurDlluRR	15	5	11.99543876	432	375
12	ullDRlldrrruUllD	17	4	10.74599498	397	349
13	rrddlUdlluRRluR	15	4	1.507203675	125	118
14	luurDDuurrdLulld drUluRR	23	6	1.142136556	96	91
15	lddrURurDDullluR R	17	6	25.53950377	627	571
16	RuLdlddrrUdlluur urrDLddlluuluRRR dLulDDldR	41	12	60.81368455	882	852
17	DDuurrdLullddrUl uRR	19	6	6.780900225	303	282
18	None	0	0	1800.308409	5490	5028
19	urrrrddllURlLuRlll dRRRuRDldR	28	10	202.0855481	1779	1673
20	urrrrddlUruulldRu rDDullllddrURRlll uRRR	38	10	1769.723384	4965	4922
21	drrURurDlldlluRR RlluRR	22	8	26.23239092	603	577
22	drruLdrddllUluRR urDlluRR	25	7	8.481752629	328	320
23	ulDLLulldRRRRDld RurU	20	10	22.84414825	581	547

24	ulDLdRdrUUllLulldRRRR	21	10	18.36280736	514	484
25	rUdlluRuRD	10	4	0.087517894	20	18
26	dlUruLdLLdlluRRRRurDllluRR	26	11	12.74842709	397	389
27	ullDRdrU	8	3	0.288918481	52	44
28	dllUUluurDldRlldRRuRDldR	24	9	39.48657924	766	704
29	urrrddlUruLdLLuurDldRRurDllldlluRRRR	36	12	37.37556776	675	667
30	ruURurDDullluRRdlddlUU	22	8	55.95081462	968	892
31	UUddrrruuullDRlldrrruUllDldR	29	7	69.24646929	1015	996
32	UruulldDuR	10	3	1.45234016	139	120
33	ullDLLulldRRRRldRRUluR	22	11	33.32259468	716	643
34	ulDLLulldRRRRDrUdlldrUluR	26	11	38.41261582	733	715
35	luulDrdLuLLulldRRRRdrUdrddllluUruLLulldRRRR	43	16	35.97178923	688	678
36	uulDLdRdrUUllLuldRRRR	22	10	5.866786221	265	250
37	DlddrrruuuLrdddlluurRurDllldrUluRRlluRR	41	9	59.7954642	916	897
38	ulDLLulldRRRRDrddlUruUlldR	26	11	25.31643163	580	570
39	drruuullDulldRRurrddd llUURurDDulllluRR	37	10	10.73624941	327	297
40	ulDLLulldRRRRlddrruUlldR	24	9	24.04385949	573	559
41	ulDrdLuLLulldRRRRdrUlllldRR	26	11	4.3921911	213	209
42	uulDLddrUluRdrUllLulldRRRR	26	10	8.694651737	304	299
43	uulDrdLuLLulldRRRRdrUllllddrrUdlluurRR	38	12	130.5211405	1340	1302
44	uulDLddrUluRdrUdldllluuRRR	26	8	30.37289353	651	631
45	uulDrdLLdlUURRdrUllllluurDldRRR	30	12	16.08633672	449	428

46	uulDrdLuLLddlluu RRRRdrUdlLullldd rrUdlluurRRR	44	14	9.963871169	325	312
47	dllURlluurrDRDLL dlU	19	8	22.54716945	606	495
48	dllURlluurrDLrRDL dlU	20	8	11.28164128	399	339
49	lddRUluurrrrdddl LruLrruullD	27	5	850.9594775	3758	3442
50	None	0	0	1800.137742	6616	4689
51	None	0	0	1800.606273	5473	5002
52	None	0	0	1800.523197	6449	4668
53	lddddRluuRDrDul uluurDDldRldR	28	8	5.132955931	234	226
54	uulldDuurrddLdL UUdddlluURuRR lDllddrruUUluRd llddRdrUUU	55	18	285.8669818	2004	1987
55	None	0	0	1800.193747	5408	5035
56	DullddrRuulDrdR RRRuurrddLLLLLrr ruurDrdL	39	14	748.2329011	3422	3309
57	None	0	0	1800.388702	6074	4894
58	None	0	0	1800.883202	6117	4924
59	None	0	0	1800.437198	5987	5255
60	rrUURurDDDLLuu llDDRdrUUdrruulL LuRR	34	17	262.680704	2040	1988
61	None	0	0	1800.464688	5925	4957

Tabla 64: Resultados Amplitud

10.3.2 Profundidad

Nivel	S_Pro	P_Pro	C_Pro	T_Pro	G_Pro	E_Pro
1	LruulllldddrUluurrrddlLLrrruullllldDrrruullDldRlDruRlluurrrrddLLLrrruullldDrrrruullDldRlluurrrrddLLrruLLullldRldRluurrrdLulldrrRlluurDDluurrrrddLLrruullllddRDuluurrrrddlLrruullldD	181	31	112.3095299	649	550
2	rruUUrrdLulDDlluR	17	6	0.436729747	52	43
3	dllUluRRDldR	12	5	3.53422438	162	153
4	lULulldRRRluR	13	6	1.176407786	105	95
5	ldlULulldRRRluRRDldR	20	9	8.048066447	275	258
6	ullldrURurDllluRR	18	5	0.550037241	58	52
7	ruullldRRlUluRRldR	19	6	1.229099084	86	79
8	lldlUruLulldRldRRlluurrDrrrddllUruLLullddrRlluurDrrDldRluullldRluurrdrrrddLruullllldRRldRluurrrddLUlldRluurRlldrR	112	23	136.8900028	1173	1101
9	ddllluuurDRlldrrruUlllddrUluRR	31	6	16.99686273	353	333
10	ddllluuurDlddrrruuLLrrddllUdrruulullDRlDrdrruuLrddlluUluRR	58	11	3.400090007	125	95
11	lddrURurDllluRR	15	5	16.68756399	348	339
12	ddllluuurDRlldrrruUllD	23	4	18.26070702	365	352
13	rrddlUdlluRRluR	15	4	0.190395876	32	26
14	luurDDluurrrdLulldrUluRR	25	6	0.553274363	55	47

15	lddrUlddrrruuLrddlllluuuRldRuRlldd rdrruuLrddllluuurDDldRR	56	10	8.196494633	223	202
16	RuLrrDllllddrrrUdllluurrurrdLLrddlllu uuRluRlddrdrruuLrddllluuurRlDlddrrruuruLLrdddllluurDldRluuuRRR	98	18	63.67956523	767	724
17	lddrUluRlddrdrruuLrddlluluurRlDD	32	6	2.709171254	128	116
18	None	0	0	1800.207671	4120	3961
19	urrrrddllULrRllulldRRRluRRDldR	30	10	32.21772504	621	602
20	drUluRRurrdLLrrddlULrruLullddRluurDRlldrUluRdRlluurrrrddLLrruullllddrUluurrrDullDRlldrUluurrdRldlluRRurrDLLrrDLLrruullldRldRluurrrrddlLrruullDlldrURlluRRRldR	163	40	1782.276134	4195	4087
21	drrURuLrrDlldlluRluRRldRRluR	28	10	11.34940609	364	347
22	ldddrrruuuLrdddllUluRRurDllluRR	31	7	13.75600865	334	322
23	ulDLLrrrdLullulldRRRRddlUruLLrrurDDllululldRRRRdrU	50	18	9.52830892	277	248
24	ulDLLrrrdLullulldRRRRdrUlddlUruLLulldRRRR	41	16	2.822628533	143	117
25	rUdlluRuRD	10	4	0.141715244	25	24
26	dlULLrrruLLdldlluRRluRdRRuLrrDllllluRRR	38	15	7.353760787	245	219
27	dlUdlluRuRlddrruLulD	20	5	0.451102899	52	49
28	ldlUUlldRRluuurDldRRDldR	24	9	13.52566957	381	360

29	urrrddlULLrrruLLdldlluRRluRdRRlluurDldRurrDLLLdlluRRRR	54	20	6.083038039	210	174
30	rrruuullDurrdLrddlllUURRlldrrruUlllDldRR	40	10	6.493351007	239	220
31	rrruuullDurrdLrddlllUURRlldrrruUlllDldR	39	9	4.866910699	201	183
32	lUdrUruullDRlD	14	5	0.130541597	32	22
33	lulDLLrrrdLullulldRRRRRlddlUrrrUllLLulldRRRRdrruLulDldRR	56	21	5.597717258	204	166
34	ulDLLrrDlululldRRRRdrUllddrUluR	32	11	37.80939005	657	637
35	luulDLLrrrdLullulldRRRRlDrrrddlllUdrrruulUllDrrrddlllUUruLLulldRRRR	67	20	6.070434259	196	163
36	uulDLLrrrddlUluRllulldRRRdrrULLLulldRRRR	40	16	2.110960871	120	94
37	DlddrrruuLruLrddddllluuRRurDllldduRluRluRRldR	44	11	50.80631434	810	793
38	ulDLLrrrddlUluRurDDllululldRRRRdrUldddrU	40	13	7.519258929	256	231
39	drruululDulldRRurrddd||UUluRddddrruuuLLdRurDllllu RRdrD	54	14	19.10785985	383	369
40	ulDLLulldRRRRlddrruUlldR	24	9	22.3139359	495	482
41	ulDLLrrrdLullulldRRRRdrUllldRR	30	11	2.215441341	127	108
42	uulDLLrrrdLullulldRRRRdrUlddlUruLLulldRRRR	42	16	2.6122101	133	111
43	uulDLLrrrdLullulldRRRRllllddrrrUrrUlLLrddllluuRRRR	50	16	40.99995144	614	580

44	uulDLrrdLdlllluuRRlldrrrruruulDllldd rrUdlluurRRlllddrr rUrULLrddlllluuRR R	70	16	18.1235284	445	408
45	uulDLdlUrrrdLLull uurDldRRRlddlUrr rUlldlUluurDldRR R	50	16	5.617620807	238	205
46	uulDLLrrrdLullddll uuRRRRdLrrUllllld drrUruLddlluuRRR R	52	16	4.446921246	178	157
47	dllURlluurrDLrRlld drUluulDrurDrDL dllURdrUluulDrdr ruLddllU	57	16	11.78891882	349	312
48	dllURlluurrDLrRDL dlU	20	8	1.210583263	94	77
49	None	0	0	1800.075878	4551	4470
50	llUdrrrrUdllllluUrD ldRluururrrDlDLru rDluulldldRRluurr rddLruulllulDrrrrd dlLrruulllIDrDrrru ullllldD	95	17	11.26938768	283	214
51	None	0	0	1800.262983	4221	4092
52	None	0	0	1801.436306	2989	2792
53	lddRlddRluurDluu urDlddrrDulldRlu uurDldR	39	8	1.672824928	120	100
54	uullllldDuurrdDuu rrddLdLdllUdrruU rruullllldDuurrrrdd ldldllUUddrruuUd ddlluuURuRRlldld ddrruuUllddRluur uRldldddrdrUUU	111	20	214.3458988	1472	1455
55	None	0	0	1801.002734	3591	3397
56	DullddrRlluurDrd RRRRuurrdLrdLLL LLrrrrruullD	43	14	626.9476137	2997	2981
57	None	0	0	1800.162253	4747	4704
58	None	0	0	1800.696165	4821	4772

59	drrRllluuurrrrddDllluuurrdRlulldddrrruUllDLruulldDrdRldlUUrrurrddLLLulDuU	75	15	89.74548749	977	939
60	rrUUllDDrruuRurDllluRRdrDDLLrruullllddRdrUUdlluurRlldrrrruuLLL	63	19	88.08072493	1083	1045
61	None	0	0	1800.367696	4341	4291

Tabla 65: Resultados Profundidad

10.3.3 Profundidad Iterativa

Nivel	S_Plt	P_Plt	C_Plt	T_Plt	G_Plt	E_Plt
1	None	0	0	2083.490831	252193	154169
2	rruUUrrdLulDDllu R	17	6	28.74760294	4880	3588
3	dllUluRRDldR	12	5	9.408098346	2658	1567
4	lULuRllldRRR	12	6	5.253758119	1403	911
5	ldlULulldRRRluRR DldR	20	9	626.8593114	88482	56920
6	ullldrURurDllluR R	18	5	4.441087175	716	556
7	ruullldRRlURlluR R	18	6	15.57166844	2612	2139
8	None	0	0	2638.868003	351075	213229
9	ddlluURlluRR	12	4	3.900334519	1066	659
10	ullDRlldrUdrruUl LulDD	22	7	667.1194829	77321	53731
11	lddrURurDllluRR	15	5	51.07917305	10504	6902
12	ullDRlldrrruUllD	17	4	65.93348544	11111	7315
13	rrddlUdlluRRluR	15	4	9.252064666	1964	1580
14	luurDDuurrdLulld drUluRR	23	6	32.87167097	4387	3914
15	lddrURurDDullluR R	17	6	158.4099635	27886	18292
16	None	0	0	2152.932041	206853	157080
17	DDuurrdLullddrUl uRR	19	6	171.9417373	24146	18739
18	None	0	0	1806.021431	260371	162355
19	None	0	0	2204.304741	210943	132988
20	None	0	0	2170.710833	242303	156469
21	drrURurDlldlluRR RlluRR	22	8	272.3941171	34011	24871
22	None	0	0	2119.160048	233009	177043
23	ulDLLulldRRRRDld RurU	20	10	299.4279711	43458	29857
24	ulDLdRdrUUllLull dRRRR	21	10	351.1528581	46728	33373
25	rUdlluRuRD	10	4	0.289619527	104	92
26	dlUruLdLLdlluRRR RurDlluRR	26	11	1866.761804	186003	138680
27	ullDRdrU	8	3	0.452170541	183	130
28	None	0	0	2081.425881	251566	173015
29	None	0	0	2436.517452	216795	151466

30	ruURurDDullluRRl dddlUU	22	8	231.0439224	29556	21985
31	ruUluRRlddd lUrdr ruuLulDDlUddR	29	9	1864.931688	166848	126666
32	UruulldRlD	10	3	2.485179204	815	553
33	lulDLLulldRRRRld RRUluR	22	11	1000.425856	126786	87049
34	None	0	0	1846.944187	189298	138290
35	None	0	0	2208.604187	169464	133586
36	uulDLdRdrUUllLul ldRRRR	22	10	72.82199657	9193	6881
37	None	0	0	1933.971154	218386	166802
38	ulDLLulldRRRRDr ddlUruUlldR	26	11	1605.147556	160038	115529
39	drruululDulldRRur rdddllUURurDDull luRR	37	10	1011.33406	65810	56290
40	ulDLLulldRRRRldd rruUlldR	24	9	975.7005398	110149	80144
41	ulDrdLuLLulldRRR RdrUlllldRR	26	11	410.8481562	40201	32098
42	uulDLLulldRRRRld drUrUdlluR	26	10	266.1361438	25860	20364
43	None	0	0	2110.659821	178680	135443
44	uulDLddlluuRRRld drUrUdlluR	26	8	1000.273899	100140	71460
45	uulDrdLLdlUURRd rUlllluurDldRRR	30	12	877.6156531	73174	59268
46	None	0	0	1868.767028	134771	116485
47	dllURlluurrDRDLL dlU	19	8	70.40455687	10410	7101
48	dllURlluurrDLrRDL dlU	20	8	24.78537461	3790	2750
49	None	0	0	2214.682422	268084	167031
50	None	0	0	2847.165255	420458	239213
51	None	0	0	2596.865144	259786	181696
52	None	0	0	2488.375511	285281	168447
53	lddddRluuRDrDull uuurDDldRldR	28	8	487.4749043	43415	33998
54	None	0	0	1945.009343	106128	84131
55	None	0	0	1873.137809	261511	172972
56	None	0	0	2093.67487	177544	134977
57	None	0	0	3132.207063	418998	253931
58	None	0	0	2993.113556	398207	242687
59	None	0	0	2648.083459	286524	186279

60	None	0	0	2524.979181	288985	205522
61	None	0	0	2657.57257	304369	205844

Tabla 66: Resultados Profundidad Iterativa

10.3.4A*

Nivel	S_ASt	P_ASt	C_ASt	T_ASt	G_ASt	E_ASt
1	None	0	0	1800.439657	7461	4492
2	rruUUrrdLulDDlluR	17	6	0.640660255	88	54
3	dllUluRRDldR	12	5	0.809477594	125	65
4	lULuRllldRRR	12	6	0.393853776	79	43
5	dllULulldRRRluRRDldR	20	9	4.008276446	297	164
6	ullldrURurDlluRR	18	5	0.526635985	68	45
7	ruullldRRlURlluRR	18	6	1.374596557	130	77
8	lldlUruLulldRRlldRRuRRlDldRR	28	11	35.97795843	1013	537
9	ddlluURlluRR	12	4	0.347841907	67	39
10	ullDDuurrdLrddlLuUluRR	22	7	15.7543747	609	400
11	lddrURurDlluRR	15	5	2.597830793	236	146
12	ullDRlldrrruUllD	17	4	2.875217923	240	148
13	rrddlUdlluRRluR	15	4	1.121910867	137	89
14	luurDDuurrdLulldrUluRR	23	6	1.195597295	119	87
15	lddrURurDDullluRR	17	6	6.751073284	400	256
16	RuLdlddrrUdlluururrDLddlluuluRRRdLulDDldR	41	12	44.98288012	950	666
17	DDuurrdLullddrUluRR	19	6	4.166550054	281	203
18	dlluRRRdrUlllUluRRRlDldRR	25	11	319.8914213	3280	1912
19	urrrrddllURlLulldRRRluRRDldR	28	10	21.59357459	737	425
20	urrrrddlUruulldRurDDullllddrURRllluRRR	38	10	1139.248365	5935	3775
21	drrURurDlldlluRRRlluRR	22	8	12.00706928	551	340
22	drruLdlldrUluRRurDlluRR	25	7	7.697919638	397	286
23	ulDLLulldRRRRDldRurU	20	10	6.915928517	412	254

24	ulDLdRdrUUllLulldRRRR	21	10	7.548715171	425	266
25	rUdlluRuRD	10	4	0.074399785	23	14
26	dlUruLdLLdlluRRRRurDllluRR	26	11	10.30354538	480	323
27	ullDRdrU	8	3	0.160910224	44	23
28	dllUUluurDldRlldRRuRDldR	24	9	14.49463956	615	369
29	urrrddlUruLdLLdlluRRRRurDllluurDldRR	36	12	33.26984182	863	584
30	ruURurDDullluRRldddlUU	22	8	15.14652271	600	413
31	UUddrrruuullDRlldrrruUllDldR	29	7	54.7457939	1151	835
32	UruulldDuR	10	3	0.492121269	96	52
33	ullDLLulldRRRRldRRUluR	22	11	4.590908562	327	183
34	ulDLLulldRRRRDrUlldddrUluR	26	11	20.71787038	701	480
35	luulDrdLuLLulldRRRRdrUdrddllluUruLLulldRRRR	43	16	33.11695704	783	620
36	uulDLdRdrUUllLuldRRRR	22	10	3.162957963	243	152
37	DlddrrruuuLrdddlluurRurDllldrUluRRlluRR	41	9	47.45991966	1032	765
38	ulDLLulldRRRRDrddlUruUlldR	26	11	16.23351533	581	404
39	drruuullDulldRRurrddllUURurDDulluRR	37	10	4.679485207	248	177
40	ulDLLulldRRRRlddrruUlldR	24	9	15.25058324	608	400
41	ulDrdLuLLulldRRRRdrUlllRR	26	11	4.223478405	263	187
42	uulDLLulldRRRRlddrUrUdlluR	26	10	6.708516717	342	235
43	uulDrdLuLLulldRRRRdrUllllddrrUdlluurRR	38	12	54.49253643	1136	782
44	uulDLddlluuRRRldrUrUdlluR	26	8	12.13185546	514	352
45	uulDrdLLdlUURRdrUlllluurDldRRR	30	12	10.82692372	471	317

46	uulDrdLuLLddlluu RRRRdrUldLullldd rrUdlluurRRR	44	14	8.723971189	356	272
47	dllURlluurrDRDLL dlU	19	8	2.23138552	208	116
48	dllURlluurrDLrRDL dlU	20	8	2.335119804	208	118
49	lddRUluurrrrddlLr dLrruuullD	27	5	134.9599802	2070	1205
50	None	0	0	1800.780078	7969	4479
51	None	0	0	1800.020265	7149	4696
52	None	0	0	1800.175863	7928	4293
53	lddddRluuRDrDull uuurDDldRldR	28	8	4.643971928	263	200
54	uulldDuurrddLdL UUdddlluURuRR lDlldddrruUUluRl dlddRdrUUU	55	18	228.2850337	2210	1701
55	None	0	0	1800.643596	7214	4791
56	DullddrRuulDrdR RRRuurrdLrdLLLLL rrrruulD	39	14	319.7537831	2954	2015
57	None	0	0	1800.105798	7643	4610
58	None	0	0	1800.053455	7590	4648
59	None	0	0	1800.029567	7436	5085
60	rrUURurDDDLLuu llDDRdrUUdrruulL LuRR	34	17	190.13338	2242	1631
61	None	0	0	1800.219416	7328	4758

*Tabla 67: Resultados A**

10.3.5IDA*

Nivel	S_IDA	P_IDA	C_IDA	T_IDA	G_IDA	E_IDA
1	None	0	0	2146.302853	256944	133111
2	rruUUrrdLulDDllu R	17	6	6.383325092	1387	871
3	dllUluRRDldR	12	5	0.501524177	205	100
4	lULuRllldRRR	12	6	0.543395429	222	121
5	ldlULulldRRRluRR DldR	20	9	11.98368991	2650	1459
6	ullldrURurDllluR R	18	5	1.337081364	285	183
7	ruullldRRlURlluR R	18	6	3.155372987	794	468
8	lldlUruLulldRldRRl uRRRlDldRR	28	11	605.6267748	87794	48982
9	ddlluURlluRR	12	4	0.561499513	197	107
10	ullDRlldrUdrruUl LulDD	22	7	105.4315372	15661	9700
11	lddrURurDllluRR	15	5	4.499110527	1311	737
12	ullDRlldrrruUllD	17	4	14.36945464	2902	1720
13	rrddlUdlluRRluR	15	4	3.092724784	923	553
14	luurDDuurrdLulld drUluRR	23	6	19.17239096	3051	2286
15	lddrURurDDullluR R	17	6	19.33427923	4342	2524
16	None	0	0	2288.059369	210252	148091
17	DDuurrdLullddrUl uRR	19	6	64.49574551	10635	7369
18	dlluRRRdrUlllUluR RRlDldRR	25	11	519.9634704	84447	44747
19	urrrrddllURlLulldR RRluRRDldR	28	10	81.33361922	10163	5778
20	None	0	0	2268.331221	241381	141144
21	drrURurDlldlluRR RlluRR	22	8	27.65959944	5014	3016
22	drruLrdddllUluRR urDllluRR	25	7	943.6302855	113875	79173
23	ulDLLulldRRRRDld RurU	20	10	29.27268362	5846	3384
24	ulDLdRdrUUllLull dRRRR	21	10	30.64599641	5900	3503
25	rUdlluRuRD	10	4	0.134340924	70	39

26	dlUruLdLLdlluRRR RurDllluRR	26	11	285.1024382	36531	23440
27	ullDRdrU	8	3	0.162879309	82	39
28	ldlUUlldRRluuurDl dRRDldR	24	9	112.39294	19723	11593
29	None	0	0	2590.956507	240575	142469
30	ruURurDDullluRRl dddlUU	22	8	39.70678154	6862	4409
31	ruUluRRlddd lUrdr ruuLulDDlUddR	29	9	487.9582402	53826	35886
32	UruulldRlD	10	3	0.512967982	246	118
33	lulDLLulldRRRRld RRUluR	22	11	23.71428171	5015	2879
34	ulDLLulldRRRRDr Ullddd rUluR	26	11	142.0081868	20720	13012
35	None	0	0	2003.167112	155242	108557
36	uulDLdRdrUUllLul ldRRRR	22	10	10.69986205	2037	1270
37	None	0	0	1963.224691	213678	152664
38	ulDLLulldRRRRDr ddlUruUlldR	26	11	123.7311232	16911	10835
39	drruululDulldRRur rdddllUURurDDull luRR	37	10	334.8708578	28066	20877
40	ulDLLulldRRRRldd rruUlldR	24	9	78.93416734	13005	7941
41	ulDrdLuLLulldRRR RdrUlldRR	26	11	92.27269867	11874	8028
42	uulDLLulldRRRRld drUrUdlluR	26	10	38.97624514	5544	3546
43	None	0	0	2269.265608	183913	132908
44	uulDLddlluuRRRld drUrUdlluR	26	8	57.25049364	8365	5375
45	uulDrdLLdlUURRd rUlllluurDldRRR	30	12	161.1661383	18807	12795
46	None	0	0	2104.257743	151069	113671
47	dllURlluurrDRDLL dlU	19	8	5.586345266	1282	709
48	dllURlluurrDLrRDL dlU	20	8	4.946275257	1074	628
49	lddRUluurrrrddlLr dLrruuullD	27	5	554.6639487	79160	43591
50	None	0	0	1934.153664	250885	133676
51	None	0	0	1991.702737	203526	132412
52	None	0	0	2082.436476	221328	126876

53	lddddRluuRDrDull uuurDDldRldR	28	8	162.2350645	16399	11449
54	None	0	0	2448.986247	122343	89225
55	None	0	0	1826.164047	249573	144996
56	None	0	0	2434.430742	199021	137662
57	None	0	0	2412.790325	291121	167723
58	None	0	0	2319.851324	278805	162120
59	None	0	0	2061.352487	237634	139788
60	None	0	0	1847.055887	206392	136995
61	None	0	0	2072.167939	213535	140040

*Tabla 68: Resultados IDA**

10.4 ANEXO IV: Manual de Usuario

En este anexo se explica en detalle el funcionamiento de los programas que componen el proyecto para que su uso sea comprensible por todo tipo de usuarios del sistema. Se procederá a describir el modo de empleo tanto del solucionador como de la interfaz gráfica.

Tanto el programa solucionador de niveles como la interfaz gráfica del juego pueden ejecutarse haciendo doble clic sobre el fichero de Python en Windows 8.1, pero se recomienda su ejecución desde consola o terminal. A continuación se procederá a explicar dónde encontrar la Línea de Comandos o Terminal en algunos de los sistemas operativos más empleados y cómo ejecutar los programas.

WINDOWS 7

En este caso la Línea de Comandos puede encontrarse de la siguiente forma:
`Inicio > Todos los Programas > Accesorios > Símbolo del Sistema.`

WINDOWS 8

Muchas de las distribuciones de Windows 8.1 instaladas de fábrica incluyen un menú de inicio similar al incluido en versiones anteriores de Windows. En estos casos se puede acceder a la consola de comandos de un modo bastante similar al empleado en el caso de Windows 7.

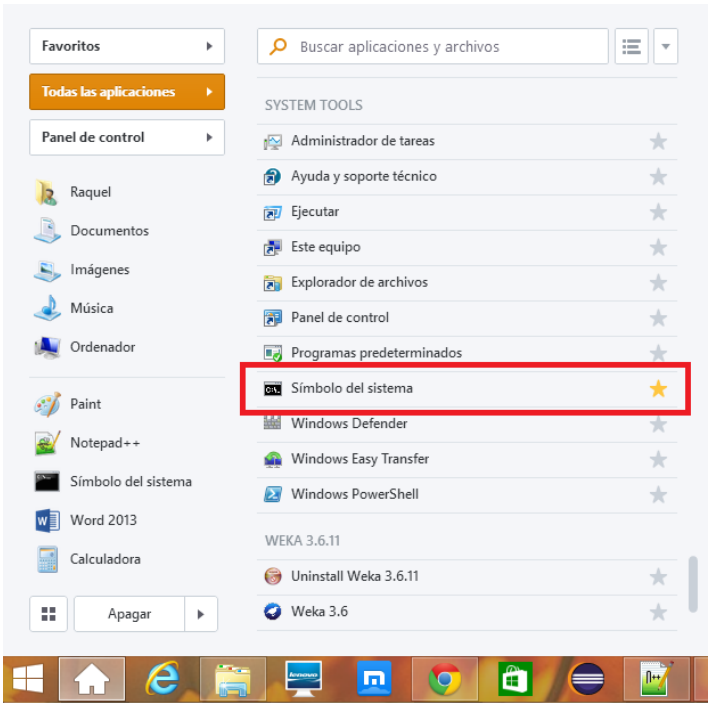

Ilustración 70: Símbolo del Sistema en Windows 8

En el caso de que Windows 8 no cuente con un menú de Inicio instalado, también es posible iniciar la Línea de Comandos desde la opción de “buscar”, que aparece desde el menú que se despliega al deslizar el puntero en la parte derecha de la pantalla, o en la lupa del menú que aparece pulsando la tecla Windows.

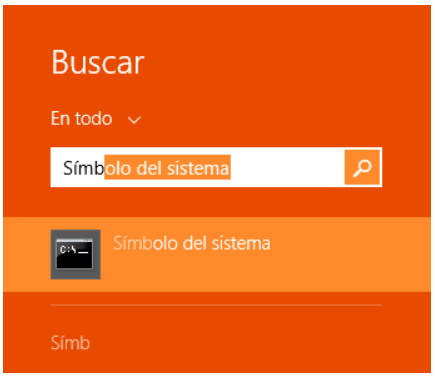

Ilustración 71: Buscar Símbolo del Sistema

LINUX

Algunas distribuciones de Linux cuentan con la terminal en el escritorio, aunque si no es así, puede encontrarse con el panel de búsqueda (Ubuntu) o en el menú de Aplicaciones (algunas distribuciones de Debian).

Ilustración 72: Terminal en Ubuntu

Ilustración 73: Terminal en Debian

Tras abrir la Ventana de Comandos o Terminal habría que desplazarse al directorio donde se encuentra el archivo Python que se desea ejecutar. Para desplazarse al directorio deseado se puede emplear "*cd RUTACOMPLETA*" o bien se van recorriendo los subniveles de uno en uno con "*cd NOMBRESUBDIRECTORIO*". Se puede volver al directorio superior con "*cd ..*" o visualizar el contenido de un directorio con *dir* (Windows) o *ls* (Linux).

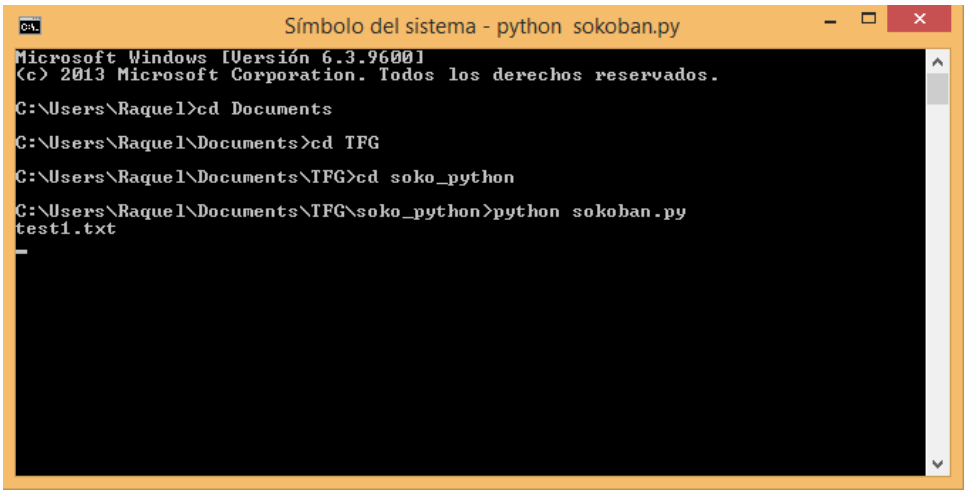

Ilustración 74: Navegación por Directorios en CMD

10.4.1 Requisitos del Sistema

Para poder ejecutar los programas de este proyecto es indispensable que el sistema a utilizar disponga de los siguientes elementos:

- **Python 2.7:** Este proyecto se ha probado en concreto con la versión 2.7.9, pero cualquier versión que sea Python 2.7 debería funcionar.
- **Pygame:** La versión a utilizar debe ser compatible con Python2. Para el desarrollo del proyecto se ha empleado concretamente es la versión 1.9.2a0, por lo que es la más recomendada, aunque versiones superiores deberían servir también.
- **Numpy:** se ha empleado la versión 1.9.1. Versiones iguales y superiores a esta deberían funcionar correctamente.

10.4.2 Ejecución del Software para resolver tableros

El programa de Python que se encarga de resolver las distintas codificaciones de tableros de Sokoban no cuenta con una interfaz gráfica, por lo que la información que ofrece será mostrada por una Línea de Comandos o Terminal y por los ficheros generados.

Una vez situados en el directorio donde se encuentra el archivo, bastaría con introducir el comando "*python sokoban.py*" para que el programa comience a ejecutar.

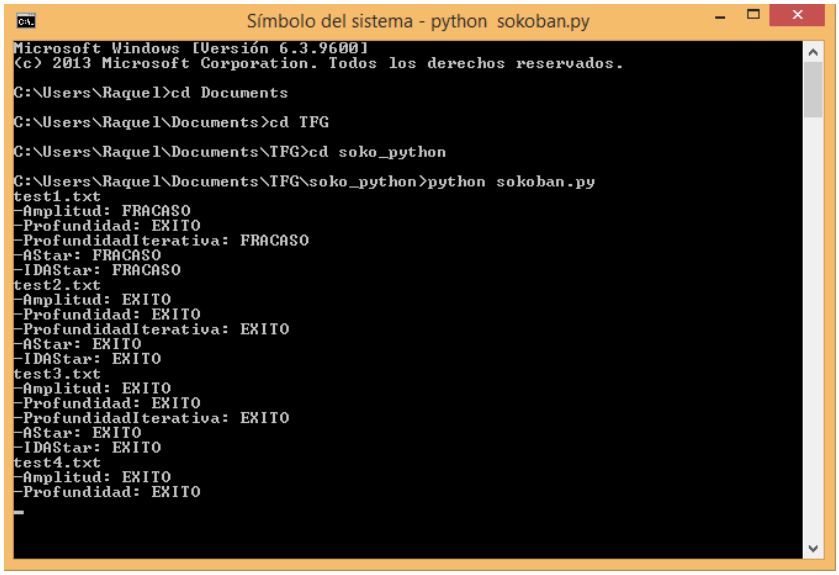

Ilustración 75: Ejecución de sokoban.py

El sistema irá informando por la consola del sistema del progreso realizado y los algoritmos que han resultado en éxito o fracaso. Cada vez que termine de ejecutar los algoritmos sobre un nivel, se guardará su correspondiente fichero con la explicación de los resultados obtenidos en la carpeta *results*. Al finalizar toda la ejecución se generarán dos ficheros más: uno llamado *resumen.txt* que contiene una salida similar a la mostrada por pantalla que incluye un recuento de soluciones encontradas por algoritmo y otro fichero de nombre *datos.txt* que contiene toda la información extraída del análisis: tiempos, nodos generados, soluciones, pasos, etc.

Es indispensable para que el programa funcione correctamente que los ficheros que contienen los niveles a resolver se encuentren en la carpeta llamada “levels” y que exista un directorio “results” donde se guardarán los ficheros generados.

NOTAS
• La ejecución del programa crea o reemplaza (en caso de ya existir) los ficheros con los datos y soluciones en el directorio “results”. Si hay datos de alguna ejecución anterior que desee conservar, debería copiar el contenido de la carpeta antes de volver a ejecutar el script.

Tabla 69: Nota Sobre ejecución del solver

10.4.3 Interfaz de Juego y Visualización de Resultados

Para ejecutar el módulo de juego y visualización de resultados, es necesario escribir en la consola de comandos o terminal la orden “*python menú.py*”. Se abrirá una ventana con la primera pantalla del menú, que muestra las opciones para Jugar, Mostrar Solución o Salir del sistema.

La Navegación por los distintos menús se realiza con las teclas de dirección del teclado y las opciones seleccionadas se aceptan con la tecla *Enter* del teclado.

En el primer menú aparece la opción *Salir* que finaliza la ejecución del programa, aunque también es posible cerrar la ventana en cualquier momento con el botón del aspa de la esquina superior derecha de la aplicación.

10.4.3.1 Jugar

A continuación se muestra la secuencia de pasos que hay que seguir para jugar a un nivel de Sokoban.

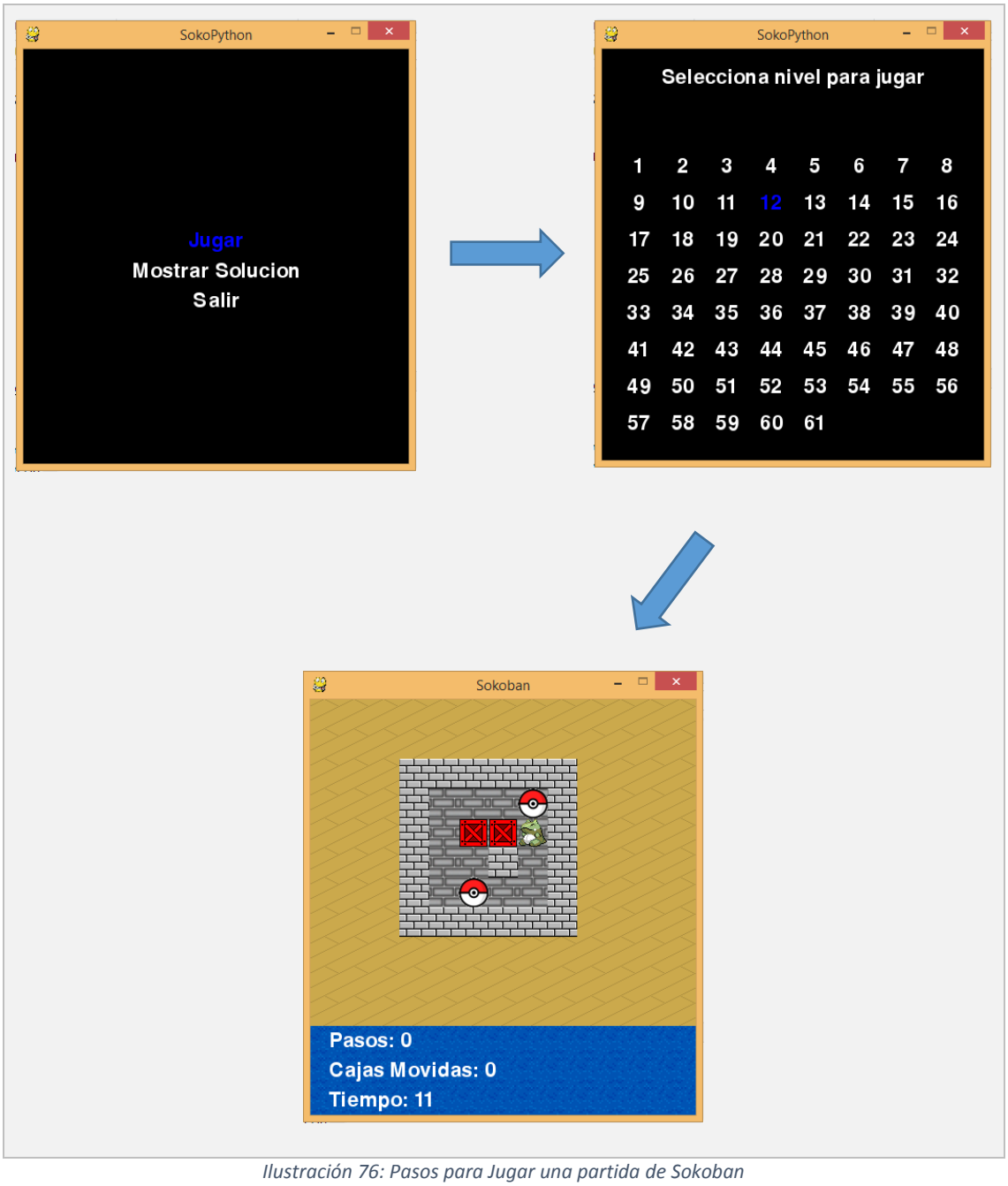

Ilustración 76: Pasos para Jugar una partida de Sokoban

Para jugar una partida, basta por tanto con indicar la opción de "*Jugar*" del menú y seleccionar el nivel que se desea emplear.

NOTAS
• Para poder jugar a cualquier nivel de Sokoban, los correspondientes ficheros del tablero deben de encontrarse en la carpeta "*levels*". En este caso el programa dará un error por consola y se cerrará.

Tabla 70: Nota sobre el modo Jugar

10.4.3.2 Mostrar Solución

Los pasos a realizar para visualizar la solución de un determinado tablero son los siguientes:

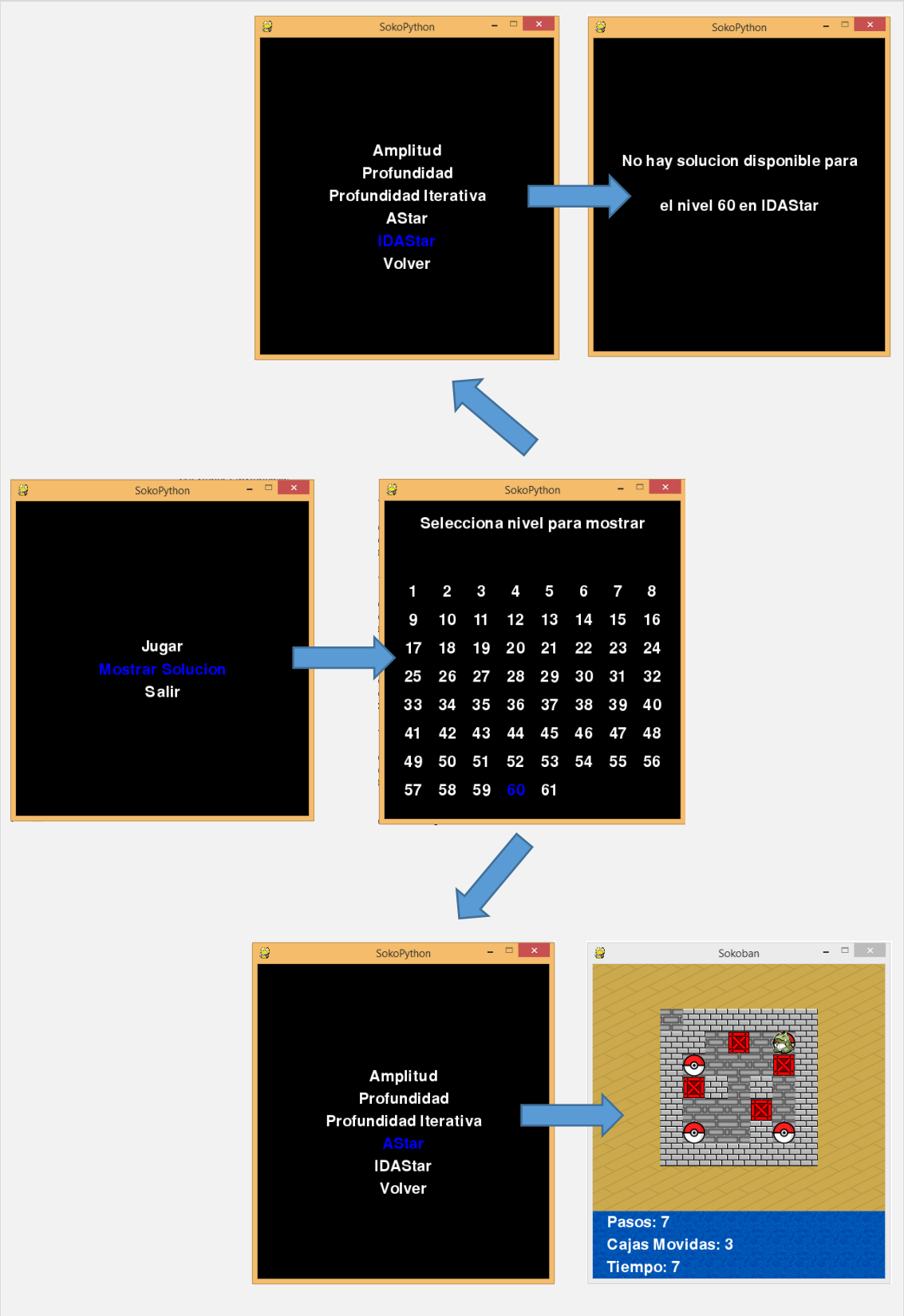

Ilustración 77: Pasos para visualizar una secuencia de solución de una instancia

Como se explica en la *Ilustración 77*, para poder visualizar una solución encontrada para un determinado nivel, hay que seleccionar la opción "*Mostrar Solucion*" del menú, indicar el número de problema a emplear y el algoritmo de búsqueda del que se desea observar la solución. Si se encontró una solución con el algoritmo escogido para el nivel seleccionado, se mostrará el nivel, que irá cambiando según la secuencia de pasos de la solución. En cambio, si no existe una solución válida, se mostrará un mensaje informativo y se volverá a la selección de algoritmo.

NOTAS
• Para poder mostrar la representación gráfica de los niveles, los correspondientes ficheros del tablero deben de encontrarse en la carpeta "*levels*". En este caso el programa dará un error por consola y se cerrará. • No es posible mostrar las soluciones de los algoritmos si los ficheros de resultados generados por el programa *sokoban.py* no se encuentran en la carpeta correcta, "*results*". En este caso el programa dará un error por consola y se cerrará.

Tabla 71: Nota sobre el modo Mostrar Solución